VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE DEFORMOVATELNÉHO POLE MARKERŮ

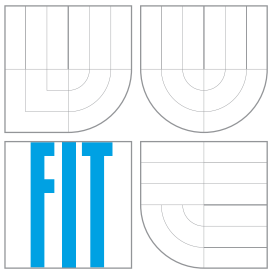
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MIROSLAV SCHERY

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE DEFORMOVATELNÉHO POLE MARKERŮ

DETECTION OF DEFORMABLE MARKER FIELD

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MIROSLAV SCHERY
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. ADAM HEROUT, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce se zabývá studiem rozšířené reality a návrhem algoritmu detektoru pro modifikovaný Uniform Marker Field odolný vůči deformaci. Součástí práce je studium existujících typů markerů. Důležitou součástí je popis Uniform Marker Field techniky, ze které pak vychází zadání modifikace určené k implementaci. Práce se věnuje také popisu architektury CUDA, na kterou je implementována první část algoritmu detektoru. Vytvořený detektor je testován na rychlost, úspěšnost detekce a odolnost vůči rozmazání.

Abstract

This Thesis is focused on study of augmented reality and creation of algorithm for a uniform marker field detector. The marker field is modified to be tolerant to a high degree of deformation. Existing marker types are studied. Important part of the paper is a description of uniform marker field technique, from which a modified assignment is derived. It also describes CUDA architecture on which the first part of the detection algorithm is implemented. Deformation tolerance, detection rate and speed tests are performed on the resulting detector algorithm.

Klíčová slova

Uniform marker field, marker, rozšířená realita, detekce hrany, paralelná implementace, CUDA, deformovatelné pole markerů

Keywords

Uniform marker field, marker, augmeted reality, edge detection, parallel implementation, CUDA, deformable marker field

Citace

Miroslav Schery: Detekce deformovatelného pole markerů, diplomová práce, Brno, FIT VUT v Brně, 2013

Detekce deformovatelného pole markerů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Doc. Ing. Adama Herouta Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Miroslav Schery
22. května 2013

Poděkování

Ďakujem vedúcemu práce docentovi Adamovi Heroutov za vedenie a cenné podnety. Mojim rodičom za podporu a bezmedznú vieru vo mňa. Svojej priateľke Klaudii za podporu a trpezlivosť. A všetkým, ktorí mi pri tvorbe tejto práce pomohli, inšpirovali a poskytli cenné rady.

Obsah

Kapitola 1

Úvod

Dostupnosť mobilných zariadení s narastajúcim výpočtovým výkonom zvyšuje záujem o vývoj nových postupov pre rozšírenú realitu. Jednou z najnovších metód určenia polohy kamery v obraze je použitie uniformného poľa markerov. Detektor vytvorený pre túto metódu dosahuje vysokého výkonu, pretože sa spolieha na planaritu markerov. Cieľom tejto práce je použiť modifikované pole markerov k vytvoreniu detektora odolného voči deformáciám. Vychádza z návrhu uniformného poľa markerov s jednotlivými bunkami tvaru šesťuholníkov, uvedeného v článku od Horváth et al. [5].

V tejto diplomovej práci sa zoznámime s pojmom rozšírená realita. Uvedieme si rôzne typy používaných markerov. Zameriame sa na preštudovanie uniformného poľa markerov a možností vytvorenia modifikácie pre podporu jeho deformovateľnosti. Oboznámime sa s platformou CUDA, ktorá bude použitá na experimentovanie s paralelnou verziou algoritmu.

Navrhnutý bude algoritmus na detekciu modifikovaného poľa markerov na CPU. Analyzujeme možnosti paralelizácie jednotlivých častí na platforme CUDA. Popísané budú implementácie algoritmu na CPU aj GPU. Výsledný detektor poľa markerov bude testovaný na rýchlosť a úspešnosť detekcie pri rôznych negatívnych vplyvoch.

Práca nadväzuje na semestrálny projekt, v ktorom bola spracovaná a popísaná problematika rozšírenej reality. Zameraný bol konkrétne na štúdium uniformného poľa markerov. Implementovaná bola malá časť prvotného návrhu algoritmu.

Kapitola 2

Rozšírená realita

Rozšírená realita (angl. Augmented reality) berie digitálne alebo počítačom generované informácie, či už sú to obrázky, audio, video a hmatové vnemy, a prekrýva ich v prostredí v reálnom čase [2]. Na rozdiel od virtuálnej reality nenahradzuje reálny svet, ale dopĺňa ho o virtuálne objekty, ktoré sú vložené do okolitého sveta. Ideálne by z pohľadu užívateľa mali virtuálne objekty koexistovať s reálnymi objektmi v priestore. Rozšírená realita môže dopĺňať vnemy pre všetkých päť zmyslov, ale v súčasnosti je najrozšírenejšie pridávanie len vizuálnej informácie.

Pravá rozšírená realita musí spĺňať tri charakteristické prvky [2]:

- kombinuje reálne a virtuálne informácie,
- je interaktívna v reálnom čase,
- pracuje s objektami v 3D, a je použitá v 3D prostredí

V súčasnosti je rozšírená realita používaná v rôznych aplikáciách a prostrediach. Navigačné aplikácie rozpoznávajú cesty a vyznačujú trasu. V turizme sa na niektorých miestach používa systém, ktorý si návštevníci nasadia na hlavu a ktorý zobrazuje pamiatky a krajinu tak, ako vyzerali v minulosti. Ďalšou z možnosti je preklad cudzojazyčných textov pri ktorých sa pomocou OCR preloží text a zobrazí namiesto pôvodného znenia.

Nasledujúce techniky tvoria jadro technológií ktoré umožňujú vytvoriť aplikácie pre rozšírenú realitu [14]:

- Sledovacia (angl. tracking) technika – slúži na získanie informácií o okolitom svete a následnú orientáciu pri pohybe.
- Interakcia a užívateľské rozhranie – slúži na vytvorenie intuitívneho ovládania pre interakciu s virtuálnym obsahom
- Zobrazovacia technika – predstavuje spôsob, akým je rozšírená realita zobrazovaná pre užívateľa

Bližšie sa budeme venovať problematike sledovania, alebo inak povedané, spracovaniu dát popisujúcich prostredie, v ktorom sa nachádza systém pre rozšírenú realitu.

2.1 Techniky sledovania prostredia

Metódy pre zber informácií o okolitom svete rozdeľujeme do troch kategórií: sledovanie pomocou senzorov, obrazové sledovanie (angl. vision-based) a hybridné sledovanie.

2.1.1 Sledovanie pomocou senzorov

Sledovanie pomocou senzorov môže zahŕňať použitie magnetického, akustického, optického, inerčného alebo mechanického senzoru. Tieto technológie boli väčšinou vyvíjané pre virtuálnu realitu, ale užitočné sú aj pre rozšírenú realitu ako zdroje doplnkových informácií, vytvárajúc tak robustnejší sledovací systém.

2.1.2 Obrazové sledovanie

Do tejto kategórie patria algoritmy z oblasti počítačového videnia, ktoré sa delia do dvoch tried. Prvá trieda je zložená z metód na extrakciu príznakov v obraze a druhá obsahuje metódy na sledovanie modelu.

Pre určenie polohy kamery v reálnom čase boli spočiatku používané metódy sledujúce umelo vytvorené markery vložené do snímaného prostredia. Viac sa markerom venuje podkapitola 2.2.

K zisteniu pozície kamery v priestore sa dá okrem markerov použiť aj množina prirodzene sa vyskytujúcich príznakov, ako sú body, čiary, hrany alebo textúry. Po spustení aplikácie sa vytvorí skupina príznakov, ktorá sa pri pohybe analyzuje a vypočíta sa tak poloha kamery zo vzájomnej zmeny pozícií. Veľkou výhodou tejto metódy je možnosť opustiť pôvodný pohľad do „scény“, pričom systém bude schopný zistiť a udržať si prehľad o pozícii a orientácii v prostredí. Dosahuje toho postupným detekovaním nových príznakov. Priebežne ich spracováva a pridáva do vnútornej reprezentácie modelu prostredia. Oproti metóde používajúcej markery, ktorá prestane fungovať, keď sa z kamery stratí marker, je z tohto pohľadu odolnejšia. Vďaka týmto vlastnostiam je v súčasnosti jednou z najaktívnejších oblastí výskumu [14].

Najnovším trendom je sledovanie s použitím modelu. Pod modelom chápeme množinu príznakov sledovaného objektu, ktorý môže byť vo formáte CAD alebo 2D šablóny. Najčastejším príznakom, na ktorého detekovanie sa sústredí najviac techník vytvárajúcich model objektu, je hrana. Detekovanie hrany nie je výpočtovo náročné a je odolná voči zmenám osvetlenia. Definícia textúry pre model objektu zvyšuje odolnosť systému a jeho presnosť pri určovaní polohy vzhľadom ku kamere.

Obrázok 2.1: Ukážka sledovania modelu (prevzaté z Vision Based 3D Tracking and Pose Estimation for Mixed Reality [1])

2.1.3 Hybridné sledovacie techniky

Tieto techniky využívajú kombináciu dát z viacerých senzorov, vrátane obrazu z kamery, k vytvoreniu aplikácií pre rozšírenú realitu tam, kde predchádzajúce postupy nestačia. V exteriéroch je napríklad GPS užitočným zdrojom informácií. Jeho použitím je aplikácia schopná pokryť väčšie okolie a naplniť ho virtuálnymi objektami, ktoré užívateľ objaví až po príchode na konkrétnu GPS súradnicu.

Problém prudkého pohybu kamerou a následného stratenia prehľadu o scéne je možné riešiť použitím akcelerometru a gyroskopov. Údaje o pohybe systému a jeho rotácií sú použité pri predikcii novej polohy objektov sledovaných v obraze.

2.2 Sledovanie s využitím markerov

Augmented Reality Group vypracovalo štúdiu pre detekciu markerov a ich dekódovanie [13]. Z tejto štúdie budú popísané niektoré systémy, ktoré používajú vizuálne markery.

Vizuálne markery sa v rozšírenej realite dodnes používajú veľmi často. Prevládajú markery v tvare štvorca, u ktorých je výhodou existencia štyroch bodov s pravidelným rozložením, vďaka ktorým sa dá presne určiť poloha kamery na základe jedného markeru. Niektoré systémy pre tento účel používajú samotné kódovanie identifikátora markeru, ktoré je rovnako perspektívne deformované.

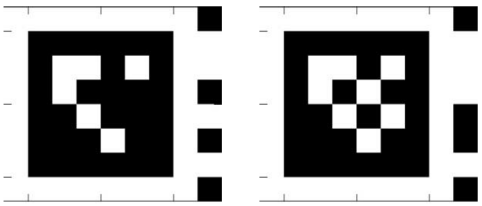
Ďalším druhom je kruhový marker, ktorý však pri sústredných neprerušovaných kružniciach poskytuje len jeden priestorový bod a preto ku kalibrácii kamery je potrebné umiestniť do priestoru minimálne tri takéto markery.

Medzi systémy používajúce vizuálne markery patria: ArToolKit[6] (volne dostupný AR toolkit), Auto-assembly[10] (AR návod na montáž zámku), Outdoor tracking, CyliCon, ArLoc, CyberCode[11] a iné.

ArToolKit používa **ATK markery**, ktoré sú zobrazené na obrázku 2.2. Marker sa skladá z vonkajšieho štvorca, v ktorom je na bielom pozadí vzor identifikujúci tento marker. Dekódovanie markeru je založené na zjednodušenom porovnávaní šablónou, ktoré porovnáva obsah markeru s vopred registrovanými vzormi v systéme.

Obrázok 2.2: ATK markery (prevzaté z [13])

HOM marker systém bol pôvodne vytvorený vo firme Siemens a slúžil pre priemyselnú dokumentáciu a aplikácie pre údržbu. Marker okrem hlavného štvorca obsahuje bočný pruh dát, vymedzený dvoma menšími čiernymi štvorcami. Umožňuje zakódovanie dodatočných 6 bitov informácie. Okrem toho zvyšuje spoľahlivosť pri rozpoznávaní. Tento systém bol v aplikáciách Siemensu využívaný často. Pôvodná verzia systému umožňovala iba spracovanie statických obrázkov, ale v súčasnosti pribudla podpora aj pre video sekvencie v reálnom čase.

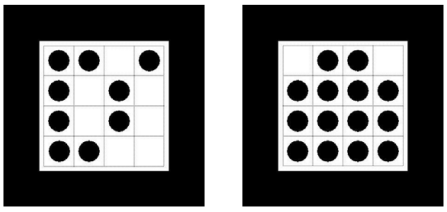


Obrázok 2.3: HOM markery (prevzaté z [13])

IGD marker systém bol vyvinutý pre nemecký výskumný projekt ARVIKA podporovaný vládou. Marker je zložený zo štvorca segmentovaného na menšie štvorce rozložené do matice rozmeru 6x6. Vnútorných 16 dlaždíc je použitých na určenie orientácie markeru a zároveň ukladajú jeho unikátny kód.

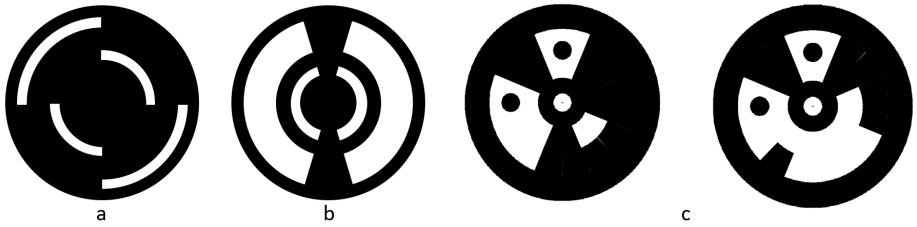
Obrázok 2.4: IGD markery (prevzaté z [13])

SCR marker systém bol vytvorený výskumnou skupinou Siemens Corporate Research (SRC). Marker je detekovaný pomocou 8 rohových bodov v obvodovom štvorci. Vnútorná časť obsahuje maticu 4x4, v ktorej je zakódovaná identifikácia markeru.



Obrázok 2.5: SCR markery (prevzaté z [13])

L. Naimark a E. Foxlin z InterSense Inc. vytvorili **Kruhový marker systém** so vstavanou dátovou maticou [7]. Tento dizajn je ich treťou iteráciou markeru pre systém používajúci kruhové markery. Prvé dve verzie mali závažný nedostatok v obmedzenej možnosti ukladať informácie. Vďaka ich jednoduchosti boli ľahko detekovateľné, avšak v prípade, keď bolo potrebné vytvoriť viacero kódov, ich marker by musel obsahovať viacero kružníc, ktoré by sa stali nečitateľné na obmedzenej ploche. Prvá a druhá verzia je na obr. 2.6 pod a) resp. b). Riešením spomenutého problému bolo implementovanie upravenej dátovej matice, ktorá je často využívaná v štvorcových markeroch.



Obrázok 2.6: Kruhové markery: (a) Prvá verzia (b) Druhá verzia (c) Tretia verzia s vyššou dátovou hustotou

Na obr. 2.6 sú pod c) zobrazené markery s kódmi 101 a 1967. Tento dizajn umožňuje 32768 rôznych kombinácií. Štruktúru tvoria tri sústredné prstence. Vonkajší prstenec je vždy čierny. Vnútorný prstenec obsahuje čierny kruh s bielym kruhom vo vnútri na rýchle objavenie stredu markeru. Prstenec nachádzajúci sa medzi nimi obsahuje dáta. Dva sústredné prstence kódujúce dáta sú rozdelené na 8 sektorov po 45 stupňoch. Tri z týchto sektorov sú použité na určenie natočenia a každý zo zvyšných 5 sektorov poskytuje priestor pre uloženie troch bitov informácie.

2.3 Uniformné pole markerov

Uniform marker field (UMF) je rovinná štruktúra, ktorej prítomnosť môže byť detekovaná v obraze a zároveň môže byť odvodená presná pozícia kamery vzhľadom na pole, od ktorejkoľvek podoblasti s definovanými minimálnymi rozmermi [12]. Spomínané podoblasti sú typom tradičného markeru. Odlišujú sa však tým, že neobsahujú geometrickú štruktúru, ktorá by slúžila výhradne na lokalizáciu markeru. Celá štruktúra je použitá ako na lokalizáciu, tak aj na kódovanie jedinečného identifikátora. V UMF sa jednotlivé markery prekrývajú, vďaka čomu je možné vypočítať pozíciu v poli, aj na základe veľmi malej oblasti nasnímanej pri veľkom priblížení. Pole markerov je uniformné v tom zmysle, že jednotlivé markery sú rovnomerne rozložené po celej ploche poľa. Z vlastností markeru zároveň vyplýva, že aj príznaky slúžiace na detekciu pozície a určenie polohy sú rovnomerne rozložené na celom UMF.

Autori konceptu UMF použili usporiadanie poľa tvaru šachovnice, teda pravidelnú mriežku vyplnenú štvorcami čiernej a bielej farby. Táto štruktúra je presne definovaná ako aperiodické binárne pole s n^2 oknami orientovateľné v štyroch smeroch. Bližšie bude táto štruktúra popísaná v nasledujúcej podkapitole.

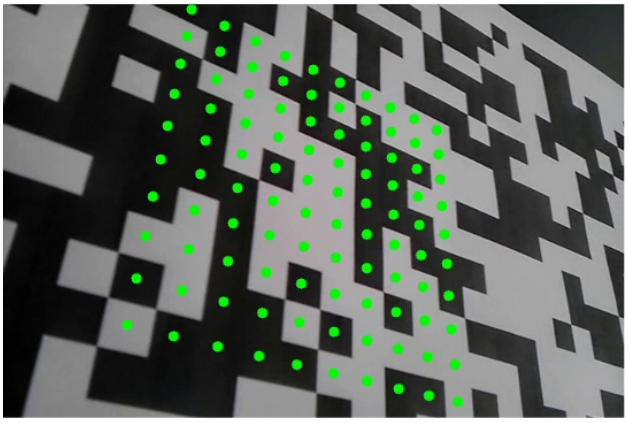
Pole s oknami rozmerov n^2

UMF pole je vytvorené tak, aby bolo možné pri natočení zistiť polohu a orientáciu aj pri zdetekovaní jediného markeru. To je možné, iba ak má pole nasledujúce vlastnosti. Každý marker je unikátny a pole je tzv. 4-orientable. Vlastnosť 4-orientable sa vzťahuje aj na jednotlivé markery a znamená, že sú unikátne pri natočení v štyroch smeroch – teda pre každých 90 stupňov. Táto vlastnosť je nutná kvôli absencii geometrického príznaku dedikovaného iba pre lokalizáciu markeru.

Analyzované boli polia s oknami, kde n nadobúdalo hodnoty 3 až 5. Aperiodické pole s oknami rozmeru $n = 3$ má obmedzenie rozmerov na maximálne 12x12 okien. Taká štruktúra nie je dostatočne veľká, aby sa výhody UMF mohli naplno využiť. V článku je navrho-

vaný ako najrozumnejší rozmer okna $n = 4$. Prezentovaným algoritmom bolo vygenerované pole o veľkosti 92x92 okien. Markery s väčšou rozlohou sú už príliš veľké pre detekovanie zblízka.

2.3.1 Detekcia šachovnicových UMF



Obrázok 2.7: UMF a zdetekované markery. (Zdroj: Uniform Marker Fields: Camera Localization By Orientable De Bruijn Tori [12])

Algoritmus pre detekciu šachovnicových UMF bol vytvorený tak, aby ho bolo možné spustiť aj na zariadeniach s obmedzenými zdrojmi, ako sú mobilné telefóny a kamery so vstavanými CPU. Pri spracovaní vstupného obrazu nepracuje so všetkými dostupnými dátami, ale používa čo najmenší počet pixelov. Algoritmus má štyri základné fázy:

1. **Extrakcia hrán** – vo vstupnom obraze sa nájde približne 100 hrán. Ako výsledok je do ďalšej fázy predaná množina hrán vyjadrená buď ako bod a smerový vektor, alebo ako dva body.

2. **Určenie dvoch úbežníkov** – (angl. vanishing point) pre dve množiny hrán, kde v množine sú približne súbežné hrany a hrany z rôznych množín sú na seba kolmé, sa určia úbežníky. Spojenie týchto dvoch bodov tvorí horizont.

3. **Vytvorenie siete nad šachovnicou** – Spojenie čiar prechádzajúcich jedným aj druhým úbežníkom vytvorí sieť, ktorá popisuje rozloženie jednotlivých dátových buniek markerov v UMF.

4. **Identifikácia markerov** – s použitím mriežky sa navzorkuje vstupný obraz a po rozpoznaní markerov je vypočítaná pozícia kamery.

Tieto fázy algoritmu budú bližšie popísané v nasledujúcich odsekoch.

Extrakcia hrán

Hľadanie hrán v obraze nie je prevádzané pre každý pixel, ale len pozdĺž tzv. skenovacích čiar v horizontálnom a vertikálnom smere. Tie sú rovnomerne rozložené cez obraz v počte okolo 10 pre jeden smer. Toto množstvo je postačujúce na extrakciu potrebného počtu hrán.

Hľadané sú hrany medzi jednotlivými štvorcami (binárnymi bunkami) markeru, keďže iba tie umožňujú lokalizáciu UMF. Moving average window detekuje body hrán p_0. Následne je aplikovaný Sobelov operátor, ktorým je získaná približná smernica priamky, na ktorej leží hrana. Podľa informácie o smere gradientu n_0 sme následne schopní určiť farbu buniek, ktoré sú hranou rozdelené.

Smernica priamky sa dodatočne spresňuje pomocou iteratívneho vyhľadávania ďalších bodov patriacich hrane v oboch smeroch. Hrana nie je zahrnutá do výslednej množiny v prípade, ak sa týmto spôsobom nenájde dostatočné množstvo a aj v prípade keď sa body odklonia od priamky s určitou toleranciou. Upresňovanie informácií je výpočetne náročnejšie, avšak so spoľahlivými údajmi postačuje nižší počet hrán.

Určenie úbežníka

Z množiny hrán sa vytvoria dva zhluky v závislosti na hodnote ich smernice. Takéto rozdelenie je na detekciu postačujúce a odpovedá dvom druhom na sebe kolmých hrán. Pre nekonzistentnejšie dáta sa odporúča použiť výber skupín pomocou metódy typu RANSAC.

Zo zhluku sa odstránia hrany, ktoré sa odchyľujú od priemernej smernice a vypočíta sa úbežník. K výpočtu použijeme homogénne súradnice pre vyjadrenie úbežníka. Čiary zhluku musia spĺňať nasledujúci predpis:

$$\forall i : v \cdot l_i = 0 \tag{2.1}$$

kde v je úbežník a l_i je i-ta hrana.

Vytvorenie siete pokrývajúcej pole markerov

S dvomi úbežníkmi vypočítanými v predchádzajúcom kroku vypočítame rovnicu priamky predstavujúcu horizont $h = v_1 \times v_2$. Čiary siete pre jeden zo smerov hrán vypočítame pomocou rovnice 2.2.

$$l_i = \hat{l}_{base} + (ki + q)\hat{h} \tag{2.2}$$

kde $\hat{x}$ je normalizovaný vektor a $\hat{l}_{base}$ je bázová čiara, ktorá prechádza cez úbežník, rôzna od horizontu. Parameter k ovláda hustotu čiar a q určuje pozíciu prvej čiary. Pre každú čiaru sa vypočíta $(ki+q)$ a nad zhlukom, ktorý takto vznikne sa pomocou lineárnej regresie nájde optimálne k a q.

Identifikácia markerov

V predchádzajúcom kroku sme získali dva zhluky čiar, ktoré vytvárajú mriežku na UMF. Tieto zhluky sa od seba líšia iba bázovou čiarou a hodnotou parametrov k a q. Body v strede buniek mriežky vypočítame podľa rovnice 2.3.

$$x_{ij} = l^{(1)}_{(i+1/2)} \times l^{(2)}_{(i+1/2)}, \forall i, j \in \mathbb{N} \tag{2.3}$$

kde $l^{(1)}$ je čiara z prvého zhluku. Z hodnôt získaných zo vstupného obrazu na pozíciách definovaných bodmi x_{ij} vznikne binárna mapa. Táto mapa po adaptívnom prahovaní odpovedá kódu zosnímanej časti UMF. Pozícia v UMF sa určí pomocou vyhľadania okien rozmeru n^2 v hash tabuľke.

2.4 Deformovateľné pole markerov

Uniformné pole markerov popísané v predchádzajúcej kapitole nepodporuje možnosť deformovať plochu, na ktorej sa nachádza. Vďaka tomu je možné implementovať veľmi rýchly detektor. Požiadavka na planárne pole je zároveň aj limitujúci faktor. Horváth et al. [5] navrhli deformovateľné pole markerov pozostávajúce zo šesťuholníkov (tvar včelieho plástu). Jednotlivé šesťuholníky sú odlíšené použitím piatich odtieňov šedej farby. Ukážka tohoto typu poľa markerov je na obrázku 2.8.

Obrázok 2.8: Pole markerov tvaru včelieho plastu s piatimi odtieňmi šedej farby.

Pole je symetrické a vytvára takzvané Y-spoje v bodoch, kde sa stretávajú hrany troch buniek. Navrhnutý bol detektor vyhľadávajúci tieto charakteristické body. Algoritmus pozostáva zo štyroch krokov:

1. **Predspracovanie vstupnej snímky** – Na vstupnú snímku sa použijú skenovacie čiary vyhľadávajúce hrany v obraze. Nad snímkou sa vytvorí hierarchická štruktúra a podľa prítomnosti hrany v bunke sa efektívne identifikujú dôležité bloky, ktoré budú ďalej spracovávané.

2. **Detekcia Y-spojov** – K detekcii Y-spojov je použitý štvorcový filter. Jeho veľkosť je daná rozmermi bunky, v ktorej sa nachádza hrana.

3. **Zostavovanie fragmentov poľa markerov** – Jednotlivé fragmenty sa zostavia prepojením Y-spojov do uniformnej hashovacej mriežky.

4. **Identifikácia markerov** – K identifikácii markeru sa používajú rozdiely intenzity farby medzi hranami v Y-spoji. Pri použití odtieňov šedej farby je odlíšiteľných šesť stavov gradientov.

Vytvorená implementácia má dostatočný výkon na spracovanie snímiek v reálnom čase aj pri rozmeroch snímku 1680x1120. Testovaním bol predvedený vyšší výkon, ako dosahuje všeobecný detektor záujmových bodov FAST. Jedným z návrhov na pokračovanie vývoja detektoru, bolo použitie samotných hrán šesťuholníka k lokalizácii markerov. Navrhovaný algoritmus detekcie markerov bude používať uvedený postup.

Kapitola 3

CUDA a paralelné programovanie

3.1 CUDA architektúra

Nasledujúca kapitola čerpá informácie z CUDA C programming guide [8].

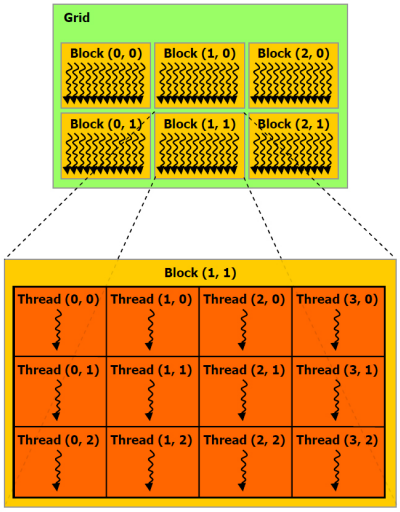
NVidia v roku 2006 vytvorila paralelnú architektúru určenú pre všeobecné výpočty s názvom CUDA (z angl. *Compute Unified Device Architecture*). Pre vývoj aplikácií na túto architektúru sa používa rozšírenie jazyka C s názvom CUDA C, čo umožňuje vývojárom používať známy programovací jazyk s nízkymi nárokmi na učenie sa nových postupov.

3.1.1 Programový model

Rozšírenie CUDA C prináša nový typ funkcie nazývaný kernel. Táto funkcia je spustená na zariadení N-krát paralelne, na rozdiel od klasickej funkcie jazyka C, ktorá je na procesore spustená jedenkrát pre jedno volanie. N rôznych CUDA vlákien vykonáva jednu inštanciu kernelu.

Rozsah paralelného spracovania je definovaný pri volaní kernelu pomocou špecifickej konfiguračnej syntaxe. Paralelná štruktúra architektúry CUDA pozostáva z jednotiek nazvaných mriežka (angl. *grid*), blok (angl. *block*) a vlákno (angl. *thread*). Elementárnym prvkom je vlákno, na ktorom je spustená jedna inštancia kernelu. Na vyššej úrovni je blok vlákien. Mriežka predstavuje najvyššiu štruktúru a skladá sa z blokov vlákien. Hierarchia štruktúr umožňuje rozdelenie mriežky a bloku vlákien do trojrozmerných podštruktúr.

Každé vlákno musí byť schopné identifikovať svoju pozíciu v spomínanej štruktúre. Využívajú sa k tomu vstavané premenné. Premenná `threadIdx` určuje polohu vlákna v bloku vlákien a `blockIdx` identifikuje pozíciu v mriežke. Obe premenné sú trojprvkové vektory. Pre rozmery bloku vlákien a mriežky existujú obmedzenia, ktoré sa líšia na základe rozdielnych verzií architektúry. Napríklad Fermi a Kepler architektúry umožňujú vytvoriť blok vlákien o maximálnej velkosti 1024. Zoznam obmedzení je uvedený v CUDA C Programming guide [8]. Obrázok 3.1 znázorňuje hierarchickú štruktúru programového modelu.



Obrázok 3.1: Hierarchická štruktúra programového modelu.

3.1.2 Pamäťový model

Pamäť z pohľadu CUDA rozdeľujeme na dve oblasti: *host* a *device*. Termín *host* sa používa na označenie systému, na ktorom je vykonávaný program v jazyku C alebo C++. Termín *device* identifikuje GPU, ktorý vzhľadom na CPU vykonáva úlohu koprocesora a na ktorom sú spúšťané kernel funkcie. Procesor nie je schopný priamo pristúpiť k dedikovanej GPU pamäti a preto CUDA umožňuje kontrolovať jej pamäťový priestor pomocou prídavných funkcií. Okrem prenosu dát medzi GPU a systémom umožňujú alokáciu a dealokáciu pamäte.

GPU zariadenie má pamäť rozdelenú na viacero typov:

- *Globálna pamäť* (angl. *Global memory*)
  Globálna pamäť je pamäť DRAM dedikovaná pre GPU. Nachádza za mimo GPU čipu a preto je prístup k jej dátam najpomalší zo všetkých typov. Avšak z pomedzi nich má najväčšiu kapacitu. Zároveň ako jediná slúži na výmenu dát so systémovou pamäťou. Všetky jej položky sú adresovateľné z každého vlákna.

- *Lokálna pamäť* (angl. *Local memory*)
  Lokálna pamäť slúži na ukladanie lokálnych premenných jednotlivých vlákien. Vlákna si navzájom nemôžu pristupovať do tohto typu pamäte. Lokálna pamäť je mapovaná do globálnej pamäte a preto by sa jej použitiu malo vyhnúť. Používa ju kompilátor v prípade keď kernel spotrebuje všetky registre.

- *Zdieľaná pamäť* (angl. *Shared memory*)
  Táto pamäť sa nachádza na čipe a preto má rýchlu odpoveď na čítanie a zápis. Jej kapacita je však obmedzená a pohybuje sa okolo 32 kB. Je určená k ukladaniu a prenosu dát v rámci bloku vlákien.

- *Registre* (angl. *Registers*)
  Na čipe sa nachádza veľké množstvo registrov, ktoré sa podľa potreby rozdelia medzi jednotlivé vlákna. Po pridelení vláknu sa stávajú privátnymi a má k nim výhradný prístup. Registe sú najrýchlejším typom pamäte.

- *Pamäť textúr* (angl. *Texture memory*) Textúry umožňujú optimalizovaný prístup do dvojrozmerného poľa. Sú definované nad globálnou pamäťou. Podporujú rôzne módy filtrovania, pričom návratový typ je obmedzený na `float`. K textúre majú prístup všetky vlákna, povolené je však iba čítanie dát.

- *Pamäť konštánt* (angl. *Constant memory*) Pamäť konštánt je určená iba na čítanie. Jej primárny účel je uchovávať konštantné dáta využívané všetkými vláknami, a preto je optimalizovaná pre broadcast dát. Dáta sú uložené v globálnej pamäti a pristupuje sa k nim cez cache.

3.1.3 Hardwarová implmentácia

Hlavným stavebným blokom CUDA architektúry sú tzv. Streaming Multiprocessor (SM). Ich počet je na GPU škálovateľný. Sú navrhnuté na vykonávanie paralelných inštrukcií na stovkách vlákien. Architektúra, ktorá to umožňuje sa nazýva SIMT (Single-Instruction, Multiple-Thread).

Úlohou multiprocesoru je riadiť sadu vlákien. Táto sada sa nazýva warp a v súčasnosti môže obsahovať 32 alebo 48 vlákien vzhľadom na verziu zariadenia. Vlákna vo warpe začínajú súčasne na rovnakej adrese inštrukcií programu. Majú však vlastný ukazovateľ na inštrukciu a registrovú sadu. To im umožňuje vykonávať inštrukcie kernelu a podmienené skoky nezávisle od ostatných vlákien, čo má za následok vynútenie serializácie pri rôznych vetvách programu a následné zníženie výkonu warpu. Súčasťou multiprocesoru je plánovač (angl. warp scheduler), ktorý mu umožňuje mať spustených viacero warpov a prepínať medzi nimi.

Pri volaní kernelu je blok vlákien rozdelený na viacero warpov a naplánuje sa jeho spustenie. Zároveň sú pre jednotlivé vlákna pridelené `threadIdx` hodnoty začínajúce od nuly.

Warp vykonáva vždy len jednu inštrukciu pre všetky vlákna, preto je najefektívnejší, keď všetky vlákna nasledujú rovnakú postupnosť inštrukcií. Vykonávanie vlákien v inej vetve programu je zatiaľ blokované.

Jednotlivé warpy nevykonávajú súčasne tú istú inštrukciu paralelne. Sú plánované a spúšťané nezávisle. K ich explicitnej synchronizácii je možné použiť CUDA funkciu, ktorá synchronizuje všetky warpy v rámci jedného bloku, avšak ďalej pokračujú v nezávislom vykonávaní programu.

Multiprocesor je schopný spustiť viacero blokov vlákien a následne medzi nimi prepínať. Umožňuje mu to efektívne sprístupniť svoje zdroje pre bloky vlákien, ktoré ich aktuálne potrebujú. Napríklad v čase, keď jeden blok čaká na načítanie dát z globálnej pamäte, multiprocesor prepne na kontext iného bloku pripraveného na výpočet. Kontexty všetkých spustených warpov sú uložené na čipe, a preto je prepínanie medzi nimi bez dodatočnej réžie.

3.1.4 Compute capability

Compute capability označuje verziu zariadenia. Skladá sa z dvoch revíznych čísiel. Na základe tohto čísla môžeme určiť generáciu zariadenia a súbor podporovaných funkcií. Prvé číslo je označenie architektúry. Aktuálne boli vydané architektúry Tesla (1), Fermi (2) a Kepler (3). Druhé číslo značí revíziu architektúry, pri ktorej zariadenie získa novú funkcionalitu. Napríklad druhá revízia Tesla architektúry (1.2) priniesla podporu atomických funkcií pre prístup do zdieľanej pamäte.

3.1.5 Optimalizačné techniky

Tvorba algoritmov pre GPU je náročnejšia ako pri programoch pre jedno jadro CPU. GPU má na rozdiel od CPU oveľa menšiu časť hardwaru venovanú riadeniu toku a zameriava sa na prácu s dátami. Preto je optimalizácia algoritmu dôležitejšia. Nasledujúci zoznam zahŕňa základné pravidlá pre tvorbu optimálneho programu pre GPU.

Riadenie toku

Rozdelenie vlákien do viacerých vetiev programu spôsobuje serializáciu a zvyšuje počet inštrukcií, ktoré musí warp vykonať. Použitiu inštrukcií if, switch, do, while a for je dobré sa vyhnúť, pokiaľ je to možné. Podmienky je dobré formulovať takým spôsobom, aby všetky vlákna v rámci jedného warpu pokračovali rovnakou vetvou. Pokiaľ sa výber vetvy podmienky líši na úrovni jednotlivých warpov, k serializácii nedôjde, pretože sa vykonávajú nezávisle.

Prístup do globálnej pamäte

Čítanie a zápis do globálnej pamäte je pomalý, pretože sa nachádza mimo čipu. Rýchlosť je naviac ovplyvnená aj použitím zarovnaného prístupu. Globálna pamäť sa skladá zo segmentov zarovnaných na 32, 64 a 128 Bajtov. Počet segmentov, ku ktorým konkrétny warp pristúpi, sa odrazí na počte potrebných pamäťových transakcií.

Použitie textúr

Pamäť textúr je optimalizovaná pre prístup k dvojrozmerným poliam. Warpy, ktoré načítavajú dáta z rovnakej oblasti textúry, majú veľkú pravdepodobnosť, že sa použije cache pamäť. Kvôli použitiu vyrovnávacej pamäte je však textúra použiteľná iba pre načítanie dát. Ak požadované položky nie sú v cache, načítajú sa z globálnej pamäti. Cache umožňuje zrýchliť načítanie dát a zároveň znížiť nároky na priepustnosť globálnej pamäti.

Pamäť konštánt

Dáta z tejto pamäte sa nachádzajú v globálnej pamäti, avšak po prvom načítaní sa uložia do cache a v prípade, že všetky vlákna warpu pristupujú k rovnakej položke, sa rýchlosť načítania vyrovná rýchlosti použitia registru. Zariadenia majú pre pamäť konštánt vyhradených 64kB.

Zdieľaná pamäť

Zdieľaná pamäť sa nachádza priamo na čipe, aby umožnila rýchly prístup k často používaným dátam v rámci jedného bloku. Môže sa tiež použiť na komunikáciu medzi jednotlivými vláknami. Ak majú bloky príliš veľké nároky na počet registrov, odporúča sa použiť zdieľanú pamäť, aby kompilátor nemusel použiť pomalú lokálnu pamäť. Pri jej správnom použití je latencia približne stokrát menšia, ako pri práci s globálnou pamäťou.

Najmenšou adresovateľnou položkou zdieľanej pamäte je *bank* o veľkosti 32 bitov. Žiadosť o prístup sa rozdelí na dve požiadavky odpovedajúce rozdeleniu warpu na dva tzv. *half-warpy*. Pre najlepšiu odozvu je potreba zaistiť, aby vlákna jedného *half-warpu* pristupovali na rôzne *banky*, alebo aby všetky čítali z jedného *banku*. Ak sa táto podmienka nedodrží, prístup všetkých vlákien sa serializuje.

Výsledok zápisu viacerých vlákien do jedného *banku* je nedefinovaný. Zápis sa podarí iba jednému vláknu. Zariadenia s compute capability 1.2 a vyššie majú k dispozícii atomické inštrukcie pre prácu s typom `int`. Pre staršie zariadenia je nutné prístup explicitne serializovať.

Prenos dát

Prenos dát medzi systémovou pamäťou a pamäťou na zariadení je taktiež časovo náročný. Jednou z možností optimalizácie je spojenie viacerých separátnych prenosov do jedého veľkého, čím sa obmedzí množstvo potrebnej réžie.

V určitých prípadoch môže byť výhodnejšie preniesť časti výpočtu na GPU, ak by prenos dát zabral viacej času ako výpočet na GPU.

CUDA podporuje asynchrónny prenos. Čas prenosu je možné úplne skryť v prípade, ak sa na GPU môžu zároveň vykonávať kernely, ktoré dané dáta nepotrebujú.

Konfigurácie kernelu

Nastavenie konfigurácie kernelu má na výsledný výkon algoritmu veľmi veľký vplyv. Inštrukcie pre jeden warp sa musia vykonávať sériovo. Multiprocesor má preto spustených viacero warpov súčasne a v čase keď sa čaká na načítanie dát a warp sa pozastaví, prepne sa kontext na iný. Vytvorenie dostatočného množstva blokov vlákien by preto malo byť jednou z najvyšších priorít. Počet spustených blokov na multiprocesore závisí od faktorov spojených s využívaním jeho zdrojov. Medzi zdroje, ktoré musí optimálny kernel využívať s mierou, patria registre a zdieľaná pamäť. Alokovanie príliš veľkého množstva jedným blokom vlákien môže zamedziť spusteniu ďalšieho bloku kvôli nedostatku zdrojov. Nový blok sa spustí, až keď sa uvoľní potrebný počet registrov alebo veľkosť zdieľanej pamäte.

3.2 Knižnica Thrust

Thrust je C++ knižnica určená pre platformu CUDA. Vychádza zo Standard Template Library (STL) [9]. Umožňuje použiť vysoko úrovňové rozhranie, ktoré je schopné spolupracovať s CUDA C. Obsahuje implementácie všeobecných paralelných algoritmov ako scan, sort a reduce. Abstrakcia na vysokej úrovni umožňuje Thrust knižnici vybrať najefektívnejšiu verziu implementácie algoritmu. Veľkou výhodou použitia tejto knižnice je dosiahnutie vysokej produktivity, robustnosti a hlavne výkonu. Navyše sa zdrojový kód stane lepšie čitateľný.

3.3 Všeobecné paralelné algoritmy

Pre určité triviálne algoritmy na CPU existujú pre paralelné zariadenia štandardné riešenia. Takisto boli navrhnuté implementácie aj pre zložitejšie algoritmy, ako je napríklad triedenie. Spomínaná knižnica Thrust implementuje tie najpoužívanejšie. Pri návrhu algoritmu budeme potrebovať implementáciu paralelnej sumy prefixov (angl. parallel prefix sum). Na jej vstupe je pole čísiel a na výstupe pole rovnakej veľkosti, pričom každá bunka obsahuje sumu všetkých čísiel s nižším indexom [3]. Existujú dve verzie: *inclusive* a *exclisive*. *Exclusive* verzia sa líši tým, že do výslednej sumy v bunke sa nezapočíta hodnota, ktorá sa v nej pôvodne nachádzala a je započítaná až v nasledujúcej bunke.

Kapitola 4

Návrh Algoritmov pre CPU a GPU

Algoritmy určené pre rozšírenú realitu sú často navrhované aj pre grafické karty za účelom využitia výpočtovej sily GPU, ktorej architektúra je stavaná k silnej paralelizácii. Tieto algoritmy sú spravidla zložitejšie na návrh. Riešenie určitého problému je často implementované najskôr pre CPU a ak je toto riešenie pomalé, analyzuje sa možnosť využitia paralelizácie. Implementácia verzie algoritmu prvotne na CPU tiež pomáha identifikovať problémy a navrhnúť optimálnejšie riešenie pred tým, než sa vývoj presunie na GPU. Súčasťou výstupu tejto diplomovej práce je CPU verzia programu, ktorá implementuje rôzne druhy optimalizácie k vytvoreniu čo najrýchlejšieho detektoru šesťuholníkov v poli markerov. Ďalšou časťou je návrh rovnakých algoritmov pre GPU. Pred implementáciou analyzujeme mieru s akou by výsledný algoritmus využíval paralelizmu a implementujeme tie, ktoré majú potenciál podstatne zrýchliť výpočet.

Definujeme si zadanie problému, ktorý musí program riešiť:

Cieľom programu je lokalizovať modifikované pole uniformných markerov, kde dátové bunky majú tvar šesťuholníka a ich farba je obmedzená na päť odtieňov šedej (viď kapitolu 2.4). Algoritmus k detekcii šesťuholníka použije aproximované hrany v obraze.

Navrhovaný algoritmus čiastočne využíva prvky algoritmov použité pre detekciu planárneho pola markerov [4]. Skladá sa z piatich základných častí:

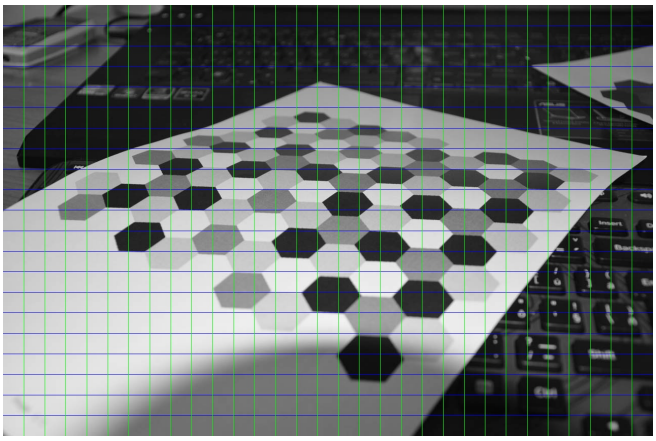
1. Detekcia hrany pomocou skenovacích čiar
2. Aproximácia hrany
3. Určenie priesečníkov
4. Výber hrán pre Y-spoj
5. Zostavenie šesťuholníkov

4.1 Verzia algoritmu pre CPU

4.1.1 Detekcia hrany pomocou skenovacích čiar

Na detekciu šesťuholníkov sa budú používať informácie o hranách v obraze. Použitie hranového detektoru nad celým obrazom je výpočtovo náročná operácia. Pri detekcii je preto

snaha vyhnúť sa častému prístupu do pamäte. Pokiaľ má hranu tvoriaca šesťuholník dostatočné rozmery na to, aby bola identifikovateľná ako súvislá čiara, postačí nájsť bod nachádzajúci sa na tejto hrane a následne dohľadať a spresniť jej polohu. K prvotnému vyhľadaniu bodov, cez ktoré prechádza hrana, sa použijú skenovacie čiary. Tie boli použité aj pri detekcii šachovnicového poľa markerov. [12].



Obrázok 4.1: Horizontálne a vertikálne skenovacie čiary.

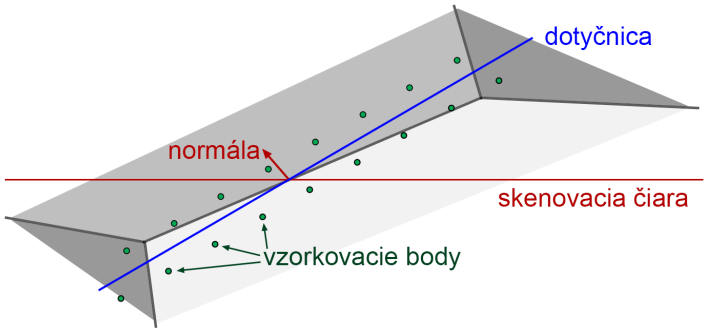
Dostatočne hustá sieť horizontálnych a vertikálnych skenovacích čiar zachytí gradienty v obraze a vygeneruje zoznam bodov určený k dohľadaniu hrán. Použitie tejto techniky dramaticky zredukuje počet načítaných pixelov. Rozostup medzi jednotlivými skenovacími čiarami však priamo ovplyvňuje veľkosť najmenšieho zdetekovateľného šesťuholníka. Aby bol šesťuholník úspešne zdetekovateľný v navrhovanom algoritme, musí byť úspešne objavených a zreťazených 5 jeho hrán. Hrany, ktoré nie sú zdetekované skenovacími čiarami v prvotnej fáze algoritmu, sa už ďalej v algoritme neobjavia. Preto je hustota mriežky kritická pre úspešnosť algoritmu. Každý bod okrem pozície v obraze bude zahŕňať aj prvotnú informáciu o normále hrany. Na výpočet normály sa použije Sobel operátor, ktorý vytvorí vektor smerujúci na stranu s vyššou intenzitou farby.

Pri detekovaní hrán môže rozmazanie obrazu, či už pohybom, alebo rozostrením spôsobiť niekoľko problémov. V algoritme môže dôjsť k nezdetekovaniu hrany v prípade, ak gradient nie je dostatočne výrazný, pretože sa rozprestiera cez viacero pixelov obrazu. Naopak, ak je gradient veľmi výrazný, ale je rozložený na viacerých pixeloch, dôjde k vygenerovaniu viacerých záznamov popisujúcich tú istú hranu. Týmto problémom sa budeme snažiť predísť pomocou postupného sčítania pozície a normály do akumulátoru v prípade, keď gradient v za sebou idúcich pixeloch indikuje prítomnosť hrany. Nakoniec akumulátor vydelíme počtom sčítaných prvkov a vygenerujeme jeden záznam kandidátneho bodu hrany.

4.1.2 Aproximácia hrany

V tejto časti algoritmu je na vstupe bod patriaci potencionálnej hrane a normála v smere intenzívnejšej farby. Získaná normála je len približná, avšak dostatočná na vygenerovanie prvých vzorkovacích bodov. Na začiatku určíme hodnoty farieb na oboch stranách hrany a s týmito hodnotami budeme následne porovnávať aproximačné vzorky. Pomocou normály vytvoríme vektor popisujúci dotyčnicu, ktorá by mala smerovať približne paralelne

s hranou. Postupným posúvaním sa popri hrane s využitím spomínaného vektora a následného odklonenia sa od dotyčnice, periodicky vzorkujeme hodnoty farieb na oboch stranách. Po každom posune po dotyčnici môžeme navzorkovať obe strany, alebo len jednu z nich a po každom ďalšom posune ich vystriedať. Výber jednej z týchto metód predstavuje výber medzi presnosťou a rýchlosťou. V navrhnutom algoritme bude uprednostnená metóda s menším počtom prístupov do pamäte a prvý spôsob vzorkovania na oboch stranách zároveň sa použije v prípade, že výsledky nebudú uspokojivé. Vzorkovanie demonštruje obrázok 4.2.



Obrázok 4.2: Demonštrácia rozmiestnenia vzoriek popri odhadovanej dotyčnici k hrane.

Z dôvodu nepresnosti prvotného odhadu normály sa po určitom posune od pôvodne zdetekovaného bodu hrany pokúsime určiť ďalší bod hrany, ktorý sa použije na spresnenie normály a dotyčnice. Tento bod vyhľadáme medzi dvomi vzorkami na opačných stranách dotyčnice.

Hodnoty farieb vzoriek sa porovnajú s farbami získanými na začiatku tejto fázy algoritmu. Pokiaľ je ich rozdiel v určitej tolerancii, môžeme predpokladať, že sa stále pohybujeme popri hrane, pričom hrana zostáva medzi náprotivnými vzorkami. Ak sa farba vzorky nezhoduje s predpokladanou farbou na danej strane dotyčnice, mohli nastať dva prípady. Buď sme narazili na koniec hrany a pravdepodobne na Y-spoj medzi šesťuholníkmi, alebo sa dotyčnica odklonila od sledovanej hrany, ktorá sa následne preťala s pomyselnou čiarou tvorenou vzorkami na jednej strane dotyčnice. V oboch situáciách sa vrátime k predchádzajúcim vzorkám, medzi ktorými vyhľadáme bod na hrane a ten uložíme ako koncový bod hrany. V tejto fáze prevedieme prvé filtrovanie hrán a pred ďalším spracovaním odstránime hrany s nulovou alebo minimálnou dĺžkou.

4.1.3 2D mriežka

Pred fázou určovania priesečníkov vytvoríme dvojrozmernú mriežku nad spracovávanou snímkou, aby sme znížili množstvo dvojíc hrán potencionálne vytvárajúcich prijateľný priesečník. Bez mriežky by zložitosť nasledujúcej fázy algoritmu bola kvadratická v závislosti na počte hrán.

Priesečník s inou hranou by sa mal nachádzať v okolí koncových bodov hrán. Pre hranu prechádzajúcu cez viacero buniek mriežky z toho vyplýva, že postačí, ak bude zaznamenaná v bunkách, v ktorých sa nachádzajú jej koncové body. Hrana teda môže byť zaregistrovaná v jednej, alebo dvoch bunkách zároveň.

Pri práci s mriežkou sa kvôli výkonu algoritmu nastavuje rozmer bunky tak, aby sa pracovalo s čo najmenej bunkami. U dvojrozmernej mriežky je to centrálna bunka a 8 okolitých buniek. V našom prípade bude vzdialenosť vychádzať z obmedzení pre priesečník, ktoré odvodíme z vlastností šesťuholníka. Hlavným faktorom je maximálna vzdialenosť koncových bodov hrán, ktorých priesečník budeme považovať za prijateľný.

4.1.4 Hľadanie priesečníkov hrán

Táto fáza algoritmu sa okrem určenia priesečníkov venuje aj ich filtrácii. Niektoré hrany je možné vyradiť ešte pred samotným výpočtom, a ďalšie možnosti filtrácie je možné vykonať po výpočte priesečníka.

Pri filtrovaní použijeme 2 hlavné druhy filtrov:

- Uhlový
- Pozičný

Nastavenie uhlového filtra bude vychádzať z vlastností uhlov v šesťuholníku, pričom sa musí počítať aj s jeho možnou deformáciou spôsobenou zmenou perspektívy a inou deformáciou plochy, na ktorej sa nachádza. Použitím tejto filtrácie priamo ovplyvňujeme schopnosť algoritmu zdetekovať deformované šesťuholníky, ale taktiež môžeme znížiť počet nesprávnych detekcií šesťuholníkov.

Základný uhlový filter, ktorý použijeme na začiatku tejto fázy, bude využívať relatívny uhol medzi hranami. Paralelné hrany majú relatívny uhol veľkosti 0 stupňov a hrany kolmé majú uhol 90 stupňov. Do úvahy sa berie menší z dvojice uhlov, ktoré vzniknú keď sa pretnú dve priamky. Tento uhol nemôže byť záporný a jeho rozsah je medzi 0 a 90 stupňov. Použitím relatívneho uhla odfiltrujeme súčasne príliš ostré, ako aj príliš tupé uhly. Ak napríklad nastavíme zahadzovanie dvojice hrán, ktorých relatívny uhol je menší ako 30 stupňov, tak prijmeme iba hrany zvierajúce uhol 30 až 150 stupňov.

K pokročilému filtrovaniu budeme potrebovať uhol, ktorý vytvárajú hrany v mieste priesečníku. V tomto prípade je výsledná hodnota uhla v intervale [0, 180] stupňov. Pokročilý filter nám umožní odfiltrovať ostré a tupé uhly nezávisle, avšak pri vyššej výpočtovej náročnosti.

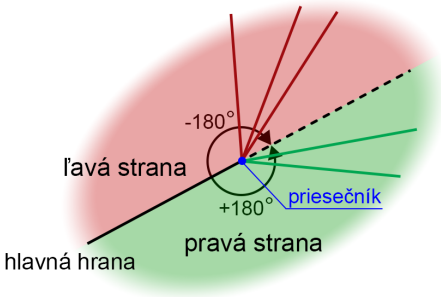
Predpoklady náročnosti na výpočet uhlových filtrov vyplývajú z rozhodnutia optimalizovať algoritmus pomocou vopred vypočítaného uhla, ktorý hrana zviera s osou x. Zníži sa tak počet volaní výpočtovo náročnej operácie na jedenkrát pre každú hranu. Na rozdiel od toho neoptimalizovaná verzia by počítala uhol medzi každými dvomi hranami.

Pri pozičnej filtrácii sú vylúčené hrany, ktoré sú príliš ďaleko od seba, alebo ktorých priesečník je príliš ďaleko od aspoň jednej z hrán.

4.1.5 Výber hrán pre Y-spoj

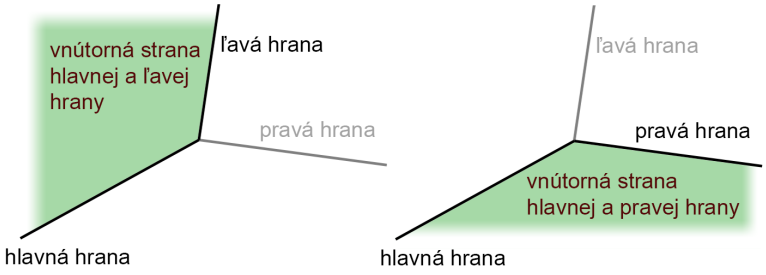
Po výpočte priesečníkov nasleduje fáza algoritmu, v ktorej sa vytvorí vyššia štruktúra prepojenia jednotlivých hrán. V pravidelnom šesťuholníkovom poli je hrana (označme si ju ako hlavná) v jednom z jej koncových bodov spojená s ďalšími dvoma hranami, čím vytvára Y-spoj (viď kapitola 2.8). Z predchádzajúcej časti algoritmu získame zoznam priesečníkov, z ktorého zostavíme výsledné Y-spoje. Prvým krokom je rozdelenie zoznamu na ľavé a pravé hrany. Toto rozdelenie je neskôr použité pri zostavení šesťuholníka. Hrany rozdelíme podľa

nasledovného pravidla. Uhol v priesečníku upravíme na interval [-180, 180] Ak je uhol záporný, hranu kategorizujeme ako ľavú, inak ako pravú. Rozdelenie demonštruje obrázok 4.3.



Obrázok 4.3: Definovanie ľavej a pravej strany pre rozdelenie hrán v Y-spoji.

Nasledujúcim krokom je výber jednej ľavej a jednej pravej hrany. Predtým, než prejdeme k výberu najvhodnejšej hrany, odstránime zo zoznamu záznamy, ktoré by v konečnom výsledku nemohli tvoriť šesťuholník. Takými záznamami sú hrany, ktorých farba sa na vnútornej strane nezhoduje. Vnútorná strana je pri spojení dvoch hrán pri uhle menšom ako 180 stupňov. Túto situáciu pre ľavú a pravú hranu znázorňuje obrázok 4.4.



Obrázok 4.4: Vnútorná strana dvojice hrán

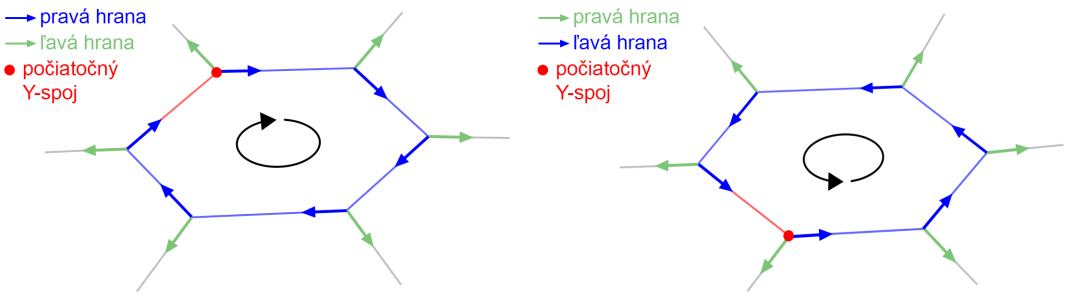
Zvyšné kandidátne hrany budú ohodnotené a vyberie sa hrana s najlepším výsledkom. Pri hodnotení sa ako kritérium použije uhol, ktorý zvierajú s hlavnou hranou a dĺžka kandidátnej hrany. Dĺžka má menšiu váhu ako uhol a do hodnotenia je zahrnutá, pretože sa predpokladá, že dlhšia kandidátna hrana presnejšie popisuje reálnu hranu z obrazu. Podľa počtu priesečníkov a ich kvality môže výsledný Y-spoj obsahovať ľavú aj pravú pripojenú hranu, iba jednu z nich, alebo žiadnu.

4.1.6 Zostavenie šesťuholníkov

Posledná fáza algoritmu má za úlohu vyhľadať postupnosť prepojených hrán a vytvoriť z nich záznam pre šesťuholník.

V ideálnom prípade by algoritmus vybral jednu z nájdených hrán a pre obe strany by si vybral určitý smer, pomocou ktorého by sa dostal do pôvodnej hrany. Spomínaný smer

môže byť proti smeru hodinových ručičiek alebo v ich smere. Ak by sme si vybrali postup v smere hodinových ručičiek, tak by sme sa pohybovali cez hrany pripojené na pravej strane Y-spoja. V opačnom smere by sme použili hrany pripojené zľava. Ako začiatok si môžeme zvoliť hociktorý z dvoch Y-spojov danej hrany. Obrázok 4.5 znázorňuje postup v smere hodinových ručičiek s prechodom cez pravé hrany v Y-spoji a proti smeru hodinových ručičiek s prechodom cez ľavé hrany.



Obrázok 4.5: Prechod cez prepojené hrany v smere hod. ručičiek cez pravé hrany (obr. vľavo) a proti smeru hod. ručičiek cez ľavé hrany (obr. vpravo).

Po šiestich iteráciách prechodu sa v ideálnom prípade vrátime do pôvodnej hrany. V reálnom prípade však takýto prípad nenastane vždy, a preto musíme vytvoriť postupy na vysporiadanie sa s nasledujúcimi problémami. Veľmi často môže dôjsť k výpadku strany šesťuholníka. Z tohto dôvodu budeme akceptovať aj postupnosť piatich hrán s tým, že sa nad nimi prevedie dodatočná kontrola. Ďalším problémom môže byť situácia, keď sa po šiestich iteráciách nevrátime do pôvodnej hrany. Preto musíme počet iterácií obmedziť na šesť, aby nemohlo dôjsť k zacykleniu. Tento prípad môže nastať u šesťuholníka, ktorý má duplicitné hrany, alebo pri sade hrán, ktoré reálne netvoria šesťuholník. Duplicitné hrany sa buď spriemerujú, alebo sa vyberie jedna z nich. Problém sady nesprávne prepojených hrán sa vyrieši v následnej filtrácii kandidátnych záznamov.

Aby sme predišli vytváraniu duplicitných záznamov toho istého šesťuholníka, musíme do Y-spoja pre pripojenú hranu vložiť informáciu o využití hrany pri konštrukcii už existujúceho záznamu. Túto premennú môžeme však nastaviť až v prípade, keď potvrdíme, že daný záznam tvorí skutočne šesťuholník.

V prípade výpadku hrany sa môžeme do Y-spoja, u ktorého k výpadku došlo, dostať už po pár iteráciách. Ak by sme túto postupnosť zahodili, mohli by sme tak zahodiť postupnosť prepojených hrán, ktorá v skutočnosti pozostáva z piatich hrán a teda by mala byť ďalej spracovaná. Preto prechod v jednom smere cez konkrétny typ hrán rozšírime o prechod v opačnom smere. Budeme sa pritom snažiť vždy nájsť šesť prepojených hrán. Ak napríklad v jednom smere nájdeme postupnosť dvoch hrán, tak v opačnom smere sa budeme snažiť nájsť zvyšné štyri. Pri zmene smeru musíme použiť hrany na opačnej strane Y-spoja.

Kandidátne záznamy sa podrobia testom, aby sa vylúčili chybné detekcie.

V prvom teste sa skontroluje paralelnosť hrán. Deformácia spôsobená perspektívnou transformáciou spôsobí, že odpovedajúce hrany už nebudú paralelné, avšak zmena uhla nie je príliš veľká. Účelom tohto filtra je odstrániť záznamy, u ktorých sada hrán tvorí nepravidelný útvar. Pri porovnávaní uhlov tak zvolíme hodnotu, ktorá bude do určitej miery povoľovať odchýlky spôsobené deformáciou pola markerov.

Druhý test odfiltruje záznamy s rôznou farbou na vnútornej strane šesťuholníka.

Záznam, ktorý prešiel filtrovaním je považovaný za šesťuholník a obsahuje dvojicu základných dát o polohe a farbe, ako aj dve polia s informáciou o hranách definujúcich jeho obvod a farbách na vonkajšej strane. Tieto polia môžu mať veľkosť 5 alebo 6 položiek na základe toho, či došlo k výpadku jednej stany alebo nie.

4.2 Verzia algoritmu pre GPU

Nasledujúci algoritmus je navrhnutý pre platformu CUDA (viď kapitolu 3.1) a preto boli pri jeho vytváraní použité určité predpoklady a špecifické vlastnosti architektúry, ktoré nemusia byť pravdivé a aplikovateľné v iných architektúrach. Použité budú niektoré zaužívané algoritmy pre riešenie štandardných problémov v paralelných aplikáciách ktoré popisuje kapitola 3.3.

Tvorba paralelnej verzie algoritmu má význam iba za určitých podmienok. Nie každý algoritmus sa dá paralelizovať. Ak by sme na GPU spustili neupravený algoritmus určený pre CPU, nedosiahli by sme žiadneho zrýchlenia. Naopak by algoritmus bežal niekoľkokrát pomalšie, pretože GPU nemá tak pokročilú riadiacu logiku ako CPU a vo všeobecnosti pracuje na 2-3 krát nižšej frekvencii. Do návrhu algoritmu by preto mala pribudnúť aj analýza miery paralelizácie, ktorá bola dosiahnutá a rozhodnutie, či má zmysel algoritmus implementovať.

Pri návrhu algoritmu pre GPU bola snaha vytvoriť ho čo najviac podobný verzii pre CPU. Niekedy však bol z optimalizačných dôvodov zvolený iný postup a preto sa výsledok niektorých fáz detekcie môže mierne líšiť.

4.2.1 Detekcia hrany pomocou skenovacích čiar

Použitie skenovacích čiar bolo zachované. Pre každý bod na tejto čiare sa vypočíta sobel operátor a následne sa použije metóda na vysporiadanie sa s rozmazaním. Implementácia tejto metódy je na CPU triviálna, avšak na GPU si vyžiadala vytvorenie špeciálneho postupu, ktorý vyprodukuje mierne odlišné výsledky. Aby sme zistili, či je v obraze rozmazaná hrana, musíme prejsť cez všetky jej pixely a rovnako ako pri CPU verzii, vytvoriť priemerné hodnoty pozície a normály. Dosiahneme toho priradením jedného vlákna na jeden pixel obrazu. Každé vlákno má za úlohu vyhľadenie hrany vľavo resp. nahor od svojej pozície. Spriemerované hodnoty uloží ako záznam hrany iba vlákno, ktorého priradený pixel je na konci detekovanej hrany. Predíde sa tak vygenerovaniu vyššieho počtu záznamov pre rozmazanú hranu.

Následne nastáva problém ako uložiť záznamy o hranách pre ďalšie fázy algoritmu. Všetky vlákna sa pokúsia uložiť dáta v rovnaký okamžik, pričom nemajú žiadnu informáciu o počte nájdených hrán zvyšnými vláknami. Nemajú tak k dispozícii spôsob, ako zistiť index na ďalšiu voľnú položku v poli určenom pre záznamy. K zoradeniu záznamov do súvislého poľa použijeme paralelnú sumu prefixov (angl. parallel prefix sum). Pomocou tejto metódy si zistíme indexy pre jednotlivé záznamy a buď vytvoríme nové zoradené pole alebo do ďalšej fázy pošleme pole indexov spolu s nezoradeným polom záznamov.

V tejto časti algoritmu môžeme predpokladať výrazné zrýchlenie výpočtu oproti CPU, pretože všetky tri podúlohy:

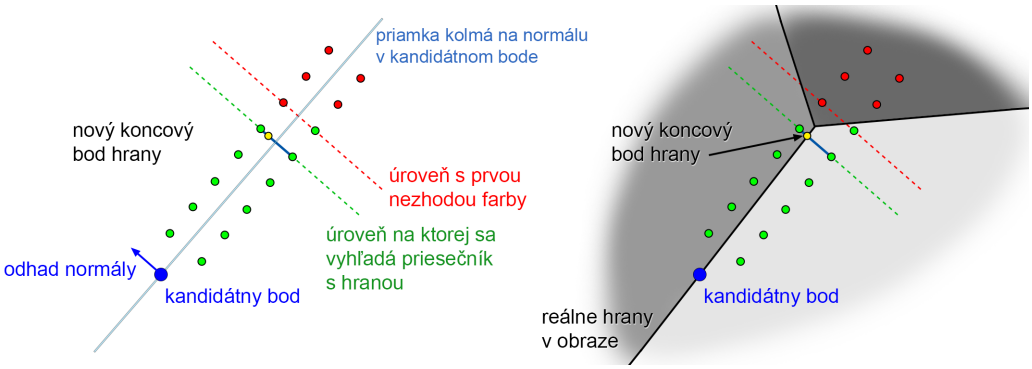
1. výpočet Sobel operátoru,
2. vysporiadanie sa s rozmazanými hranami a

3. uloženie výsledku do súvislého poľa

sú do podstatnej miery paralelizovateľné.

4.2.2 Aproximácia hrany

Na vstupe tejto fázy je zoznam bodov, v ktorých bol dostatočne veľký gradient a pravdepodobne cez ne prechádza hrana. Navyše máme k dispozícii prvotnú normálu kandidátnej hrany. Podobne ako pre CPU verziu sa na aproximáciu hrany použijú vzorkovacie body. Najskôr si zistíme farby na oboch stranách hrany a vektor približnej dotyčnice. Jednotlivé vzorkovacie body budú vygenerované paralelne, jeden bod na jednom vlákne. Každé vlákno z obrazu načíta hodnotu farby na pozícii svojho vzorkovacieho bodu a porovná ju s farbou, ktorá značí pokračovanie hrany popri dotyčnici. Ďalším postupom je určenie úrovne, kde sa po prvýkrát nezhoduje farba vo vzorkovacom bode a o jednu úroveň nižšie sa medzi náprotivnými bodmi vyhľadá priesečník s hranou. Výstupom aproximácie hrany je dvojica koncových bodov hrany. Na obrázku 4.6 je vľavo zobrazený postup algoritmu a vpravo je pridaná informácia o stave obrazu, s ktorým algoritmus pracoval.



Obrázok 4.6: Vľavo: Algoritmus GPU Aproximácie hrany, Vpravo: Pridaná informácia o obraze.

Ako dobre paralelizovateľné časti algoritmu aproximácie hrany môžeme označiť:

- výpočet pozície vzoriek,
- načítanie a porovnanie farby vo vzorkovacích bodoch a
- aproximácia koncového bodu hrany.

Naopak časti, ktoré nie je možné paralelizovať alebo pri ktorých nie sú plne využívané vlákna sú:

- inicializácia dotyčnice a načítanie farieb na stranách hrany a
- určenie úrovne, na ktorej sa prestali zhodovať farby strán.

Navrhnutý algoritmus má potenciál urýchliť výpočet oproti CPU verzii, a preto sa ho pokúsime implementovať.

4.2.3 Určenie priesečníkov

Na vstupe tejto fáze algoritmu je súvislé pole hrán. K optimálnemu výberu dvojice hrán je potrebné vytvoriť mriežku podobne ako pre CPU verziu algoritmu. K tomu účelu môžeme využiť paralelný radiaci algoritmus z knižnice `Thrust`. Pole by bolo radené pomocou kľúčov vygenerovaných pre každú hranu podľa jej príslušnosti do bunky mriežky. Miernou komplikáciou je nutnosť zaistiť, aby mohla byť hrana v mriežke zaregistrovaná v dvoch bunkách, v závislosti od pozície jej dvoch koncových bodov.

Použitím mriežky rozdelíme hrany do skupín rôznych veľkostí. Ak by sme priradili jeden blok vlákien na jednu bunku, nezaručíme efektívne využitie všetkých vlákien. Problém môže spôsobovať buď malé množstvo hrán na jeden blok vlákien, alebo bude blok preťažený vysokým počtom hrán. V tomto prípade by bolo potrebné navrhnúť pokročilé vyrovnávanie záťaže medzi blokmi. Tým by sme ale len pridali ďalšiu položku znižujúcu výkon algoritmu.

Druhou možnosťou je priradiť každému bloku jednu hranu. Aj v tomto prípade je problémom nerovnomerné zaťaženie jednotlivých blokov. Počet susedných hrán, ktoré potenciálne vytvoria prijateľný priesečník je príliš náhodný.

Tretím navrhovaným postupom je použitie dvoch kernelov. Prvý vygeneruje dvojice hrán s použitím mriežky a druhý tak bude môcť spracovávať súvislé pole týchto dvojíc. Aj keď je druhý kernel veľmi efektívny, kde jedno vlákno by mohlo pracovať s jednou dvojicou hrán, prvá časť algoritmu trpí rovnakým problémom ako predchádzajúce návrhy. Spojenie dvojíc do súvislého zoznamu by si vyžiadalo použitie ďalšej sumy prefixov.

Navrhnuté boli tri možnosti implementácie určenia priesečníkov a vo všetkých prípadoch nebolo možné dosiahnuť podstatnej paralelizácie výpočtu.

Kapitola 5

Implementácia

5.1 Detektor markerov na CPU

CPU verzia programu bola implementovaná v jazyku C++. Bol vytvorený projekt v Microsoft Visual Studiu 2010. K načítaniu a ukladaniu video súborov a obrazových dát bola použitá knižnica OpenCV verzie 2.4. Pri načítaní sa zároveň používa aj na prevod obrazu do jednokanáloveho obrazu v odtieni šedej o veľkosti 8 bit na pixel.

Zdrojový kód aplikácie je rozdelený do troch tried. Trieda `Image` sa stará o obrazové dáta. Ukladá vstupný obraz a generuje ladiace informácie do prídavných okien. Obsah týchto okien umožňuje s použitím knižnice OpenCV ukladať ako obrázky a zobrazovať v samostatných oknách na obrazovku. Ďalšia trieda `Grid` sa stará o alokovanie pamäte pre uloženie dát spojených s detekciou. Implementuje mriežkovú štruktúru pre ukladanie hrán. Posledná trieda `CPUDetector` zapuzdruje všetky algoritmy použité k detekcii šesťuholníkov. Každá jej inštancia pracuje s jedným objektom `Image` a jedným objektom `Grid`.

5.1.1 Uloženie dát

Alokácia dát so sebou prináša určité množstvo réžie. Pri malom počte volaní to nie je problém, avšak pri každom novom snímku by sa v navrhnutej aplikácii alokovali tisíce položiek pre hrany a koncové body. Čas potrebný pre alokáciu by sa mohol dostať do hodnôt, pri ktorých by sa aplikácia viditeľne spomalila. Z toho dôvodu bolo v aplikácii implementovaných niekoľko polí, ktoré sa alokujú pri vytvorení objektu triedy `Grid` s konštantnou veľkosťou. Použitie konštantne veľkého poľa pre uloženie dát vedie k:

1. vymedzeniu maximálneho počtu spracovávaných objektov v rámci jedného snímku
2. vymedzeniu maximálneho času vykonávania algoritmu.

Experimentovaním s rôzne veľkými a komplexnými snímkami sa dospeje k výberu optimálnej veľkosti týchto polí, aby sa aj v snímkach náročných na spracovanie nezahodilo veľké množstvo dát.

Trieda `Grid` obsahuje funkcie na správu týchto polí a uchováva informácie o aktuálnych indexoch na voľné bunky poľa.

Jednotlivé fázy algoritmu z kapitoly návrhu nie sú implementované ako samostatné celky. Niektoré sú zreťazené v jednej funkcii a priamo si predávajú medzivýsledky, vďaka čomu nie je potreba ukladať si globálne dáta, ktoré sa využijú iba raz v nasledujúcej fáze algoritmu. Takýmto spôsobom sú zreťazené prvé tri časti navrhovaného algoritmu:

- detekcia hrany skenovacími čiarami,
- aproximácia hrany a
- uloženie do mriežky.

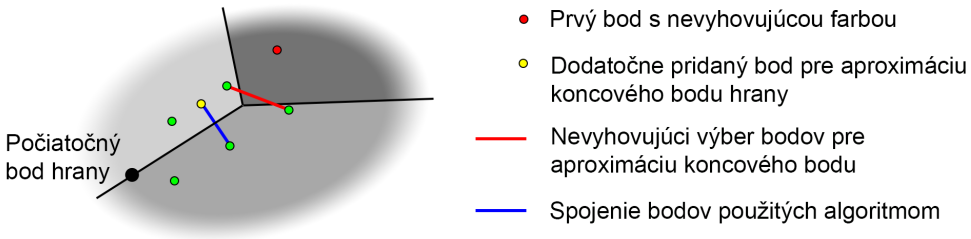
Nasleduje vyhľadanie priesečníkov spojené s výberom hrán pre Y-spoj. Poslednou samostatnou fázou je zostavenie šesťuholníkov.

5.1.2 Detekcia a aproximácia hrán

Úlohou tejto časti algoritmu je vyhľadať, aproximovať a uložiť hrany do mriežky. K vyhľadaniu hrán boli vytvorené dve funkcie. Jedna detekuje hrany na horizontálnych skenovacích čiarach a druhá na vertikálnych. Ich implementácia je vo všeobecnosti rovnaká. Líšia sa v spôsobe akým používajú sobelov operátor na určenie gradientu. Funkcia pohybujúca sa po vertikálnych skenovacích čiarach najskôr zisťuje gradient na ose y a ak je dostatočne vysoký, vypočíta aj gradient na ose x, aby zistila prvotnú normálu. Druhá funkcia má opačné poradie učenia gradientu. Tento postup bol implementovaný so zámerom znížiť počet prístupov do pamäte. Zároveň bolo snahou vyhnúť sa generovaniu veľkého počtu detekcií hrany, ktorá je paralelná so skenovacou čiarou a nachádza sa priamo na nej. Funkcie sčítajú pozície a normály susediacich pixeloch, v ktorých bol detekovaný vysoký gradient. Do vetvy algoritmu, kde sa volá funkcia aproximácie hrany sa pokračuje až na konci oblasti s vysokým gradientom. Následne sa vypočíta pozícia stredu oblasti a priemerná normála hrany.

K aproximácii hrany dochádza dvomi volaniami jednej funkcie, ktorej cieľom je určenie vždy jedného koncového bodu hrany. Pomocou parametru funkcie sa rozhodne smer, v ktorom sa bude koncový bod hľadať. V mieste kde sa detekovala hrana sa z obrazu zistia farby na oboch stranách hrany. Vypočítaný je kolmý vektor na normálu hrany a postupne sa v cykle inkrementuje. V každej iterácii sa zväčší o dopredu danú konštantnú veľkosť a pričítaním alebo odčítaním vektoru normály sa určí vzorkovací bod. Dĺžky vektorov, s ktorými sa pracuje, sú nastaviteľné pomocou dopredu definovaných parametrov. Predvolenou hodnotou je šesť pixelov. Ak sa farba vo vzorkovacom bode líši od farby, ktorá bola detekovaná na rovnakej strane hrany, končíme cyklus.

Koncový bod hrany vyhľadáme medzi dvomi vzorkovacími bodmi na opačných stranách hrany. Zahodia sa posledné tri vzorkovacie body. Prvý z nich je mimo hrany a ďalšie dva sa nepoužijú, pretože sa ich spojenie môže pretnúť s vedľajším šesťuholníkom. Táto situácia je zobrazená na obrázku 5.1. Pozícia koncového bodu hrany sa vyhľadá pomocou polenia intervalu, pričom hľadáme farbu odpovedajúcu farbe v bode prvotnej detekcie hrany.



Obrázok 5.1: Výber vzorkovacích bodov pre aproximáciu koncového bodu hrany.

Informácia o polohe sa uloží do jednej z dvoj štruktúr popisujúcich koncové body pripojené k danej hrane.

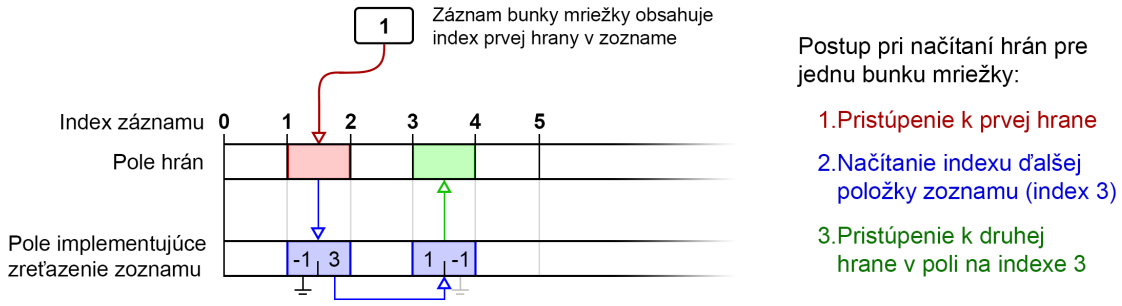
V prípade nálezu aspoň jedného koncového bodu sa informácie o hrane uložia do poľa a zavolá sa funkcia, ktorá ju zaregistruje do mriežky.

Vloženie hrany do mriežky

Implementáciu dvojrozmernej mriežky pre hrany ovplyvnilo niekoľko faktorov. Každá bunka mriežky musí umožňovať uloženie vopred neznámeho počtu hrán. Táto hodnota je zhora ohraničená, pretože v aplikácii je použité pole hrán konštantnej veľkosti a navyše každá hrana môže byť v bunke registrovaná iba raz. Zo spomínaných vlastností môžeme tiež odvodiť maximálny počet hrán uložených v celej mriežke.

Výsledná implementácia mriežky sa skladá z dvoch štruktúr. Základom je pole, ktorého počet záznamov odpovedá počtu buniek v mriežke. Záznam v tomto poli uchováva index na prvú položku zoznamu hrán zaregistrovaných ku konkrétnej bunke mriežky. Použité bolo pole štruktúr, slúžiace na vytvorenie obojsmerného zoznamu nad hranami. Toto pole má rovnakú veľkosť ako pole hrán a jeho jedna položka obsahuje dvojicu celých čísiel, ktoré slúžia ako indexy na prvky v tom istom poli.

Záznam na indexe 1 s dvojicou čísiel (-1, 3) znamená, že hrana na indexe 1 je prvým záznamom niektorej bunky mriežky a nasledujúca hrana v obojsmernom zozname je na indexe 3. Tento príklad znázorňuje obrázok 5.2.



Obrázok 5.2: Ilustrácia štruktúry a použitia mriežky pre zoskupenie hrán podľa pozície v obraze.

5.1.3 Hľadanie priesečníkov hrán a vytvorenie Y-spoja

Pri vyhľadaní priesečníkov sa iteruje medzi všetkými bunkami mriežky a vytvárajú sa dvojice hrán, ktoré sa nachádzajú v rovnakej bunke a jej susedných bunkách. Vonkajší cyklus iteruje medzi bunkami mriežky a vnútorný cyklus prechádza jednotlivé hrany v bunke. V tomto vnútornom cykle sa volá funkcia na výber hrán pre Y-spoj. Zoznam kandidátnych hrán sa tvorí výpočtom a filtrovaním priesečníkov okolitých hrán.

Na základe uhla medzi dvomi hranami dochádza k dvom druhom filtrácie. Jeho výpočet je ale veľmi náročná operácia používajúca funkciu `acos()`. Jeho počítaniu pre každú dvojicu hrán sa vyhneme pomocou výpočtu uhla s osou x jedenkrát pre každú detekovanú hranu. K tomuto výpočtu dochádza pri ukladaní hrany do poľa v predchádzajúcej časti algoritmu. Následne sa uhol medzi hranami určí s použitím predpočítaných hodnôt.

Prvá filtrácia určí relatívny uhol medzi hranami a porovná ho s minimálnou hodnotou. Táto hodnota je prednastavená na 30 stupňov. Tým sa odfiltrujú hrany, ktoré zvierajú uhol menší ako 30 stupňov alebo väčší ako 150 stupňov. Pre postupujúce hrany sa následne vyhľadajú koncové body hrán, ktoré majú medzi sebou najmenšiu vzdialenosť a tá sa porovná s maximálnou povolenou hodnotou. Predvolená je na 30 pixelov.

Tretia filtrácia využíva informáciu o uhle v koncových bodoch, ktoré boli vybraté v predchádzajúcej časti. S ich použitím sa vypočíta uhol zvieraný v mieste priesečníku a odfiltrujú sa hrany zvierajúce príliš ostrý uhol. V tomto prípade nastavený limit neovplyvňuje maximálny povolený uhol. Predvolený je uhol 80 stupňov.

V tomto štádiu dochádza k samotnému výpočtu pozície priesečníku a aj poslednej filtrácii. Kontroluje sa vzdialenosť medzi koncovými bodmi a priesečníkom. Predvolená je hodnota 25 pixelov.

Postupujúca kandidátna hrana sa uloží do zoznamu a určí sa strana, na ktorej sa nachádza (viď obr. 4.3). Po spracovaní všetkých dvojíc pre jednu hranu sa zavolá funkcia na vytvorenie Y-spoja.

Pri výbere najvhodnejšej hrany sa najskôr overí správnosť farby na strane, kde sa má nachádzať šesťuholník. Následne môže nastať pre obe strany Y-spoja jedna z nasledujúcich situácií:

- Pre konkrétnu stranu nebola nájdená vhodná hrana. V koncovom bode sa ukazovateľ na pripojenú hranu nastaví na `NULL`.
- Nájdená bola jedna hrana. Uloží sa do koncového bodu.
- V zozname sa nachádza viac ako jedna kandidátna hrana. K výberu najvhodnejšej hrany sa použije hodnotiaca funkcia. Najväčší vplyv na ohodnotenie má hodnota uhla medzi hranami. Pri uhle 120 stupňov je ohodnotenie 60 a s každým zmenšením alebo zväčšením o jeden stupeň sa znižuje o jednotku. Ďalším faktorom ovplyvňujúcim výber hrany je jej dĺžka. Prednastavené je ohodnotenie počítané ako polovica dĺžky hrany v pixeloch.

Po výbere hrán pre Y-spoj sa aktualizuje pozícia koncového bodu. Ak bola pridaná ľavá aj pravá hrana, tak sa navyše vypočíta priesečník medzi týmito hranami a pozície priesečníkov sa spriemerujú.

5.1.4 Zostavenie šesťuholníkov

K zostaveniu šesťuholníka dochádza postupným prechodom cez Y-spoje vždy v jednom smere. V algoritme sa v cykle vyberie hrana a pre oba Y-spoje sa pokúsime vyhľadať postupnosť hrán pripojených z ľavej strany. Ak týmto spôsobom nevytvoríme postupnosť šiestich rán, obrátime sa do druhého Y-spoja hrany a otočíme aj stranu na vyhľadávanie prepojených hrán. Ak nájdeme šesť alebo v horšom prípade päť prepojených hrán, skontrolujeme paralelnosť náprotivných strán. Výsledný záznam o šesťuholníku je šestica obsahujúca zoznam prepojených hrán, zoznam prvých Y-spojov, zoznam farieb na vonkajšej strane hrán, farbu šesťuholníka, počet prepojených hrán a približnú pozíciu stredu.

5.2 Detektor markerov na CUDA

Detektor markerov na architektúre CUDA bol vytvorený ako nový projekt v MS Visual Studio 2010 a bol pridaný do tzv. *Solution*. Obe triedy `Image` aj `Grid` zostali nezmenené. Bola

vytvorená nová trieda `GPUDetector`, ktorá dedí z `CPUDetector` a pridáva verzie funkcií vykonávaných na GPU. Implementácie funkcií nastavujúcich konfiguráciu kernelov a samotné kernely sa nachádzajú v súbore `CUDA.cu`.

Pred použitím funkcií pracujúcich na GPU je nutné zavolať funkciu `initializeGPU()`. Alokuje sa v nej pamäť na grafickej karte a vytvorí sa textúra pre snímku.

Pridaná bola metóda `findEdges_CUDA()`, ktorá rozdeľuje prácu na GPU do troch funkcií. Prvá funkcia detekuje hrany s použitím skenovacích čiar. Využíva k tomu dva kernely odlišujúce sa v použití vertikálnych a horizontálnych skenovacích čiar. Aj keď do veľkej miery vykonávajú rovnaké operácie, ich spojenie do jedného kernelu by pridalo viac vetvení spracovania. Taktiež by sa zhoršila čitateľnosť zdrojového kódu.

Konfigurácia kernelu nastavuje 128 vlákien na jeden blok vlákien. Bloky sú vytvárané tak, aby na jeden pixel skenovacej čiary pripadlo jedno vlákno. Každá skenovacia čiara sa začína novým blokom, preto je najefektívnejšie spracovanie obrazu, ktorého rozmery sú násobkom čísla 128.

Na začiatku si vlákno zistí svoju pozíciu v rámci konfigurácie kernelu a načíta hodnoty farieb v okolí svojej pozície v obraze do zdieľanej pamäte. Pri výpočte sobel operátora sa používajú farby pixelov v matici o rozmeroch 3x3 pixely, pričom tieto matice sa prekrývajú s maticami susedných vlákien. Načítaním potrebných dát do zdieľanej pamäte ušetríme dve tretiny prístupov do globálnej pamäte.

Po výpočte gradientu je pre jednotlivé pixely potrebné implementovať generovanie jediného záznamu pre rozmazané hrany. Implementovaný bol postup, kde každé vlákno, na ktoré pripadá posledný pixel rozmazanej hrany, vygeneruje záznam o nájdenej hrane. Vlákna začnú spracovávať informácie o gradientoch o niekoľko pixelov doľava, resp. nahor (prednastavená bola hodnota 8) od ich pôvodnej pozície. V cykle sa postupne presúvajú na pôvodný pixel a pritom si uchovávajú informácie o poslednej súvislej oblasti s vysokým gradientom. Ak sa táto oblasť končí na pozícii kam sa vrátilo vlákno, vypočíta sa pozícia stredu oblasti a priemerná normála. Tieto údaje sa uložia do zdieľanej pamäte ako kandidátna hrana. Dáta sa uložia do poľa na rovnakú pozíciu ako index vlákna, aby nedošlo ku kolízii pri zápise do rovnakej pamäťovej bunky. Vznikne tak z väčšej časti prázdne pole. K zoskupeniu záznamov o kandidátnych hranách bola implementovaná suma prefixov. Do nového poľa celých čísiel v zdieľanej pamäti zapíšu vlákna číslo 1 v prípade, ak vytvorili záznam o bode hrany, inak zapíšu číslo 0. Sumou prefixov nad týmto poľom zistíme, koľko záznamov kandidátnych bodov bolo vytvorených vláknami s nižším indexom. Tento údaj je použitý ako index do poľa. Výstupom každého bloku vlákien je súvislé pole kandidátnych bodov o veľkosti 32 záznamov, ktoré sa uložia do globálnej pamäte. Toto číslo bolo považované za dostatočne veľké pre 128 spracovávaných pixelov.

Výsledky predchádzajúceho kernelu sú uložené v jednom poli, ale záznamy s kandidátnymi bodmi v ňom nie sú súvislé. K ich zoskupeniu bola použitá ďalšia suma prefixov implementovaná v knižnici `Thrust` a kernel presúvajúci záznamy s použitím získaných indexov.

Do fázy aproximácie hrán vstupuje súvislé pole kandidátnych bodov. K spracovaniu jedného bodu je vytvorený blok s počtom 32 vlákien.

Na každé vlákno pripadá spracovanie jedného vzorkovacieho bodu. Vlákna 0-15 pracujú so vzorkovacími bodmi na svetlejšej strane hrany a vlákna 16-31 na tmavšej strane. Po výpočte pozície bodu sa s použitím textúry načíta farba, ktorá sa porovná na zhodu s farbou strany. Koncové body boli spracované paralelne, preto nevieme, kde došlo k prvej nezhode farieb. Výpočet minima bol čiastočne paralelizovateľný.

K vyhľadaniu pozície hrany medzi dvomi vzorkovacími bodmi sa použilo 16 vlákien.

Vlákna rovnomerne navzorkujú farbu snímky medzi bodmi a do zdieľanej premennej pripočítajú pozíciu v prípade, keď je farba zhodná s farbou v strede hrany. K zápisu do zdieľanej premennej je použitá atomická funkcia `atomicAdd()`. Podľa počtu zápisov sa hodnota v premennej spriemeruje. Výsledkom je pozícia jedného z koncových bodov. Určenie druhého bodu prebiehalo paralelne s prvým.

5.3 Zobrazovanie ladiacich informácií

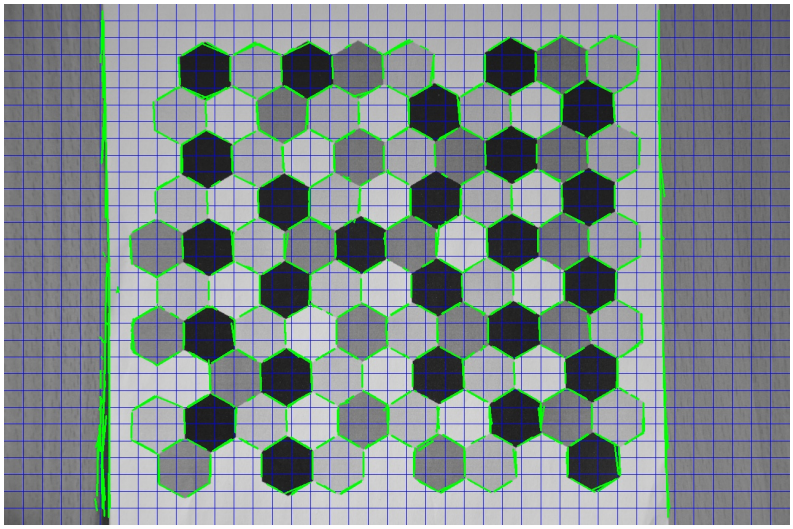
Aplikácia si vytvára štyri kópie vstupného obrazu, do ktorých generuje rozdielne ladiace informácie. Na základe nich je možné analyzovať priebeh jednotlivých fáz algoritmu.

Vykresľovanie základnej geometrie pomocou OpenCV je pomalé, čo je spôsobené aj veľkým množstvom prvkov, ktoré je potrebné vykresliť na jednotlivé snímky. Vytváranie ladiacich snímiek musí byť oddelené od časti výpočtu algoritmu, pretože sa meria ich čas výpočtu. Nie vždy je to jednoduché a preto v zdrojovom kóde existuje druhá verzia funkcie na výpočet priesečníkov, ktorá okrem pôvodného výpočtu pridáva vykresľovanie ladiacich informácií.

Prvý obrázok je využívaný pre prvé dve fázy algoritmu: detekcia a aproximácia hrán. Zobrazuje dva druhy informácie:

- sieť skenovacích čiar a
- zdetekované hrany

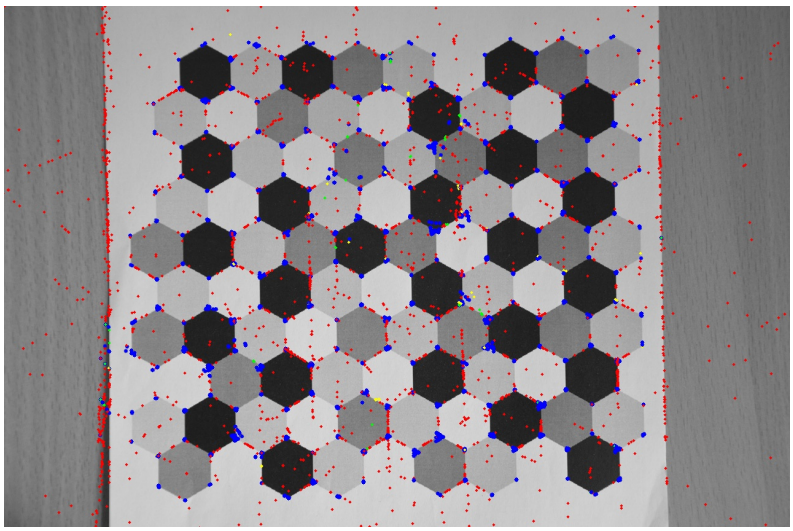
Skenovacie čiary sú vykreslené na obrázku 5.3 modrou a zdetekované hrany zelenou farbou.



Obrázok 5.3: Snímok s ladiacimi informáciami pre fázu detekcie a aproximácie hrán.

Druhý obrázok obsahuje informácie o postupe filtrovania priesečníkov. Vo väčšine prípadov je počet priesečníkov na jeden snímok rádovo v tisícoch. Pri takom množstve je nutné vykresliť priesečníky hrán vo veľkosti približne dva pixely, aby bol obrázok prehľadnejší. Do snímky sa vykresľujú štyri rôzne typy bodov s rozdielnymi farbami:

- *Červené* – Tieto priesečníky sú odfiltrované ako prvé na základe relatívneho uhla medzi hranami a v algoritme k výpočtu ich priesečníku nedôjde. K ich výpočtu dochádza iba pre účely testovania a čas výpočtu nie je zahrnutý do štatistík.
- *Žlté* – Ďalším filtrovaním sú odstránené priesečníky, ktoré sú od koncových bodov hrán ďalej, ako určitá konštantná vzdialenosť.
- *Zelené* – Druhé uhlové filtrovanie odstraňuje priesečníky, ktorých hrany tvoria v tomto bode príliš ostrý uhol.
- *Modré* – Tieto priesečníky sa dostali na koniec fázy filtrovania a budú využité pre určenie Y-spoja.

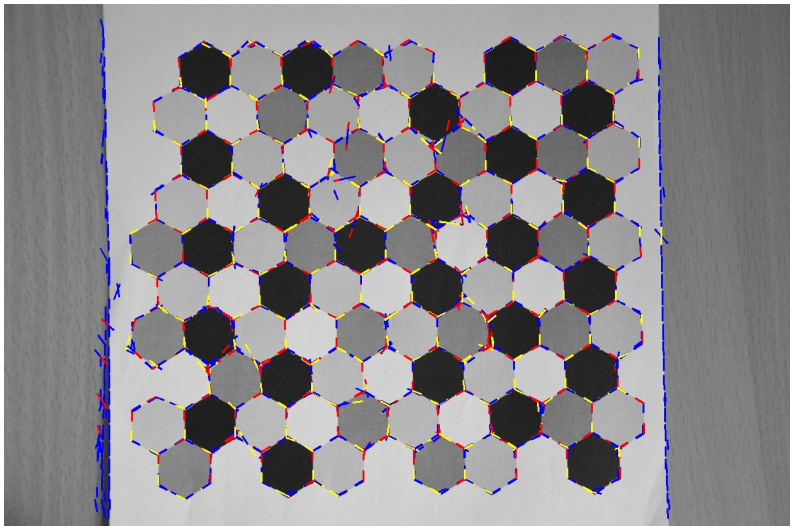


Obrázok 5.4: Snímok s ladiacimi informáciami pre fázu filtrovania priesečníkov.

Tretí snímok je využitý na zobrazovanie Y-spojov. Každý spoj sa skladá z maximálne troch hrán, ale môže nastať situácia, kedy sa zo zoznamu vhodných kandidátov nevyberie ani jeden a zostane len pôvodná hrana. Hrany v Y-spoji sú farebne rozlíšené na:

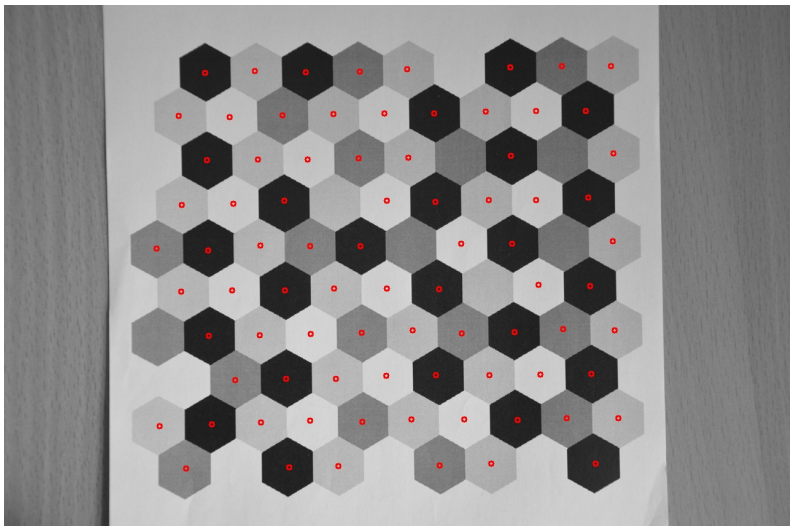
- *Modré* – Pôvodné hrany pre ktoré sa Y-spoj vytvára.
- *Červené* – Ľavé pripojené hrany.
- *Žlté* – Pravé pripojené hrany.

Vďaka farebnému rozdeleniu môžeme kontrolovať správnosť rozdeľovania pripojených hrán na pravé a ľavé.



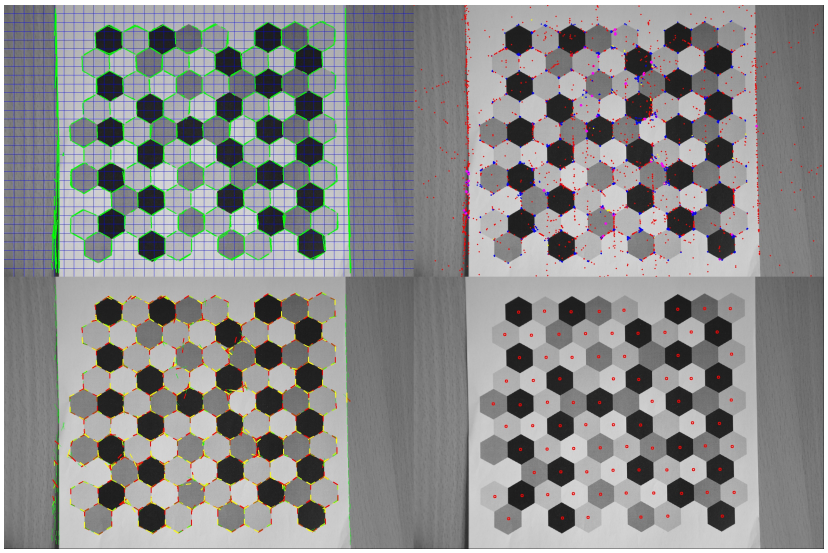
Obrázok 5.5: Snímka s ladiacimi informáciami pre fázu určenia Y-spojov.

Na poslednej samostatnej snímke sa vykresľujú body do predpokladaného stredu šesťuholníka. Bod, ktorý sa zobrazí, je červený kruh a do stredu sa vykreslí ďalší bod menších rozmerov vo farbe detekovaného šesťuholníka.



Obrázok 5.6: Snímka s ladiacimi informáciami zobrazujúca detekované šesťuholníky.

Program taktiež umožňuje vytvoriť snímku, v ktorej spojí všetky štyri predchádzajúce ladiace obrázky. Taktiež je možné uložiť ju ako `jpg` obrázok alebo `avi` video. Jej použitie je veľmi užitočné pri ladení konkrétnych snímiek videa, pretože vidíme postupné aplikovanie všetkých fáz algoritmu.



Obrázok 5.7: Snímka spájajúca predchádzajúce obrázky s ladiacimi informáciami.

Kapitola 6

Testovanie a hodnotenie

Všetky testy boli vykonávané na notebooku s procesorom Intel Core i7 Q740 1.73GHz s DDR3 pamäťou a grafickou kartou NVidia GTX 460M.

Pri implementácii boli pre ladiace účely použité postupy využívajúce vykresľovanie základných geometrických objektov do okna OpenCV knižnice. Táto metóda ladenia bola pre účely testovania rozšírená a umožňuje vykresľovať informácie do viacerých okien. Program umožňuje exportovať videá a obrázky s ladiacimi informáciami vo formáte *avi* a *jpg*.

Pre účely testovania bolo potrebné vytvoriť podporu exportovania štatistických dát. Implementovaný bol export do textového súboru, kde sú jednotlivé typy dát oddelené čiarkou a dáta pre rôzne snímky videa sú na novom riadku. Výsledný súbor je ľahko importovateľný do tabuľkového procesora.

6.1 Nastavenie konštánt detektora

Vytvorená aplikácia obsahuje desiatky konštánt, ktoré ovládajú počet použitých skenovacích čiar, rozmery mriežky pre ukladanie hrán, nastavenia filtrov a iné. V tejto časti testovania zistíme, aký vplyv majú rôzne nastavenia niektorých z konštánt.

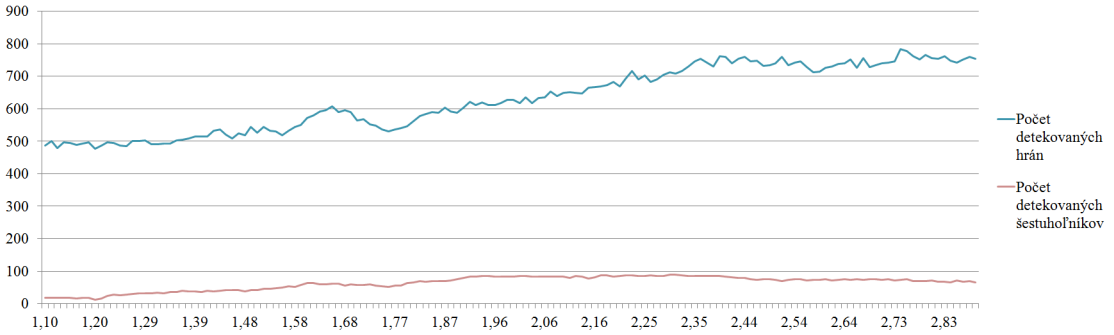
Hustota skenovacích čiar

Na počte skenovacích čiar priamo závisí schopnosť algoritmu zdetekovať dostatočné množstvo hrán pre vyhľadanie šesťuholníka. Ďalším faktorom, ktorý musíme brať do úvahy je plocha, akú šesťuholník zaberá v obraze. Pre jednoduchosť budeme počítať iba s jedným rozmerom a to s osou x. Vzdialenosti budeme uvádzať v relatívnych percentuálnych rozmeroch, kde 100% je nastavených na šírku snímky. V tom prípade 24 vertikálnych čiar rozdelí snímku na 25 blokov a navzorkuje ju každé 4% šírky obrazu. Aby sme skenovacími čiarami zdetekovali všetky hrany šesťuholníka, jeho paralelné hrany musia byť vzájomne vzdialené viac, ako je vzdialenosť dvoch blokov medzi skenovacími čiarami. Pre obraz šírky 1280 pixelov to predstavuje približne 107 pixelov. Ak zvýšime počet skenovacích čiar na 40, minimálna vzdialenosť sa zníži na 62 pixelov.

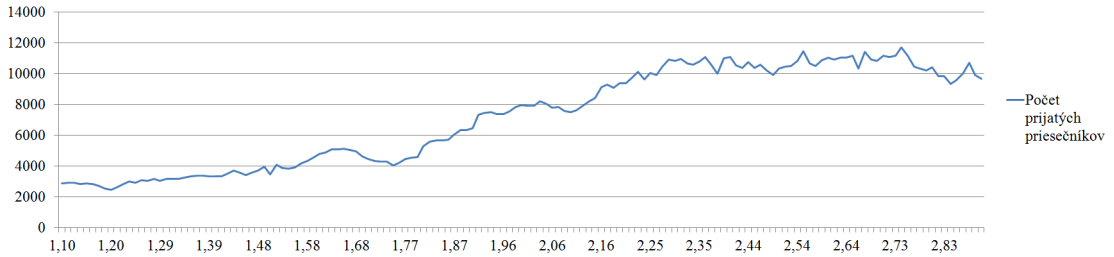
Ak do úvahy zoberieme aj druhú dimenziu s horizontálnymi skenovacími čiarami, v prípade nesplnenia minimálneho rozmeru pre dostatočné navzorkovanie v jednom rozmere, je stále veľká pravdepodobnosť, že zvyšné hrany budú zdetekované v druhom rozmere. V tomto teste budeme skúmať vplyv hustoty skenovacích čiar na úspešnosť detekcie hrán, priesečníkov a šesťuholníkov, vzhľadom na rozmery šesťuholníkov v obraze.

Jedným zo spôsobov prevedenia testu je použitie jedného obrázku a postupného zvyšovania hustoty skenovacích čiar. Druhou možnosťou je použitie konštantnej hustoty mriežky s postupným zväčšovaním rozmeru šesťuholníkov. Použitá bude druhá metóda, pretože pri nej môžeme použiť video s plynulým približovaním.

Test bol prevedený nad videom `zoom_HD.avi`. Použité boli snímky 15 až 165, počas ktorých sa šírka šesťuholníka vzhľadom na šírku bloku medzi vertikálnymi skenovacími čiarami menila v intervale hodnôt [1,1;2,9]. Počet týchto čiar bol 40.



Obrázok 6.1: Graf počtu detekovaných hrán a šesťuholníkov v závislosti na veľkosti šesťuholníka. Jednotka horizontálnej osi je šírka šesťuholníka vzhľadom na šírku bloku medzi vertikálnymi skenovacími čiarami.



Obrázok 6.2: Graf počtu detekovaných priesečníkov v závislosti na veľkosti šesťuholníka. Jednotka horizontálnej osi je šírka šesťuholníka vzhľadom na šírku bloku medzi vertikálnymi skenovacími čiarami.

V oboch grafoch 6.1 a 6.2 je vidieť postupný nárast počtu priesečníkov, hrán aj šesťuholníkov. Množstvo detekcií hrán a priesečníkov čiastočne narastá aj po prekročení hodnoty 2,0 vďaka tomu, že pole šesťuholníkov zaberá väčšiu plochu v snímke. Počet detekovaných šesťuholníkov však dosahuje vrchol práve na hodnote *2,0*, čo potvrdzuje predpoklad o minimálnej hustote skenovacích čiar.

S využitím tejto informácie môžeme nastaviť hustotu skenovacích čiar na hodnotu, ktorá bude najoptimálnejšia pre štandardnú vzdialenosť poľa markerov od kamery. Za snímku v štandardnej vzdiaľenosti od poľa markerov môžeme považovať snímku na obrázoku 6.3.

Obrázok 6.3: Snímka s poľom markerov v štandardnej vzdialenosti od kamery.

Šírka šesťuhoľníka najďalej od kamery zaberá 3,7% šírky snímku (na ose x). Aby sme dosiahli pomeru šírky šesťuholníka k šírke bloku medzi skenovacími čiarami v hodnote 2,0, musíme použiť 54 vertikálnych skenovacích čiar a 31 horizontálnych.

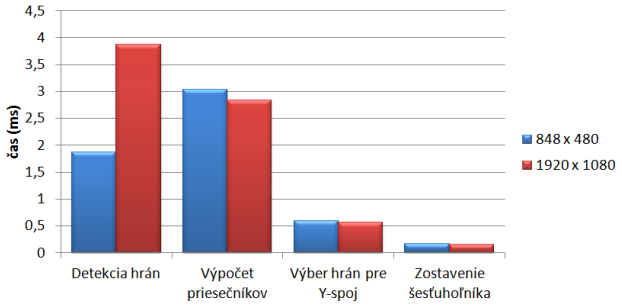
Tento test preukázal dôležitosť dostatočnej hustoty skenovacích čiar pre úspešnú detekciu. Prínosným rozšírením v tejto oblasti by bolo vytvorenie modulu pre dynamickú zmenu tejto hustoty. Okrem celkovej zmeny počtu skenovacích čiar pre celú snímku by sa hustota upravovala aj pre konkrétne oblasti v obraze.

6.2 Testy rýchlosti výpočtu algoritmu

Algoritmus bol vytvorený so zámerom vytvoriť rýchly detektor. Nasledujúce testy skúmajú rýchlosť jednotlivých fáz algoritmu, ako aj čas výpočtu pri rôznej zložitosti obrazu.

Test vplyvu veľkosti snímky

Prvým testom rýchlosti bude meranie časov potrebných pre výpočet jednotlivých fáz algoritmu. Začneme videom `rotation_SD` a `rotation_FullHD`. Porovnáme časy ovplyvnené veľkosťou vstupnej snímky. Video `rotation` má približne rovnakú zložitosť obrazu počas všetkých 261 snímiek preto môžeme hodnoty z jednotlivých snímiek spriemerovať. Výsledky tohto testu sú v grafe 6.4.



Obrázok 6.4: Graf času výpočtu jednotlivých fáz algoritmu v závislosti na rozlíšení vstupnej snímky.

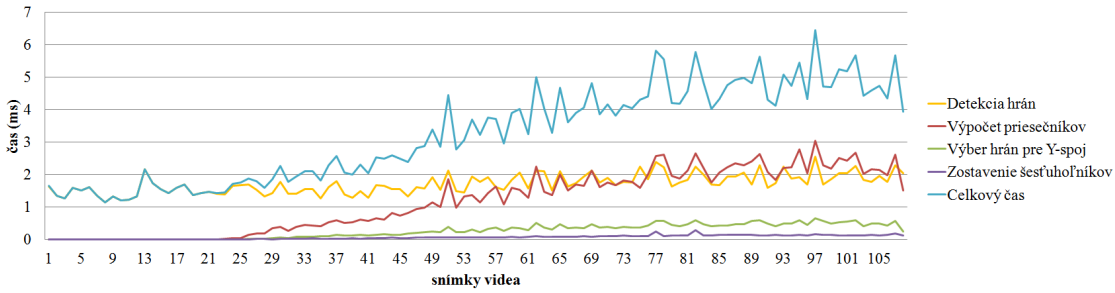
Z grafu 6.4 môžeme vyvodiť dva závery:

- Zmena rozlíšenia má silný vplyv iba na fázu detekcie hrán. Čas vykonávania tejto fázy sa zväčšuje lineárne so zväčšovaním rozmerov snímky a nie so zväčšovaním plochy snímky. Zvyšné časti algoritmu nie sú ovplyvnené veľkosťou snímky, pretože pracujú s podobnou sadou hrán, ktorá je výstupom prvej fázy a do obrázku zasahujú minimálne.
- Najviac náročná je fáza detekcie hrán a výpočtu priesečníkov. Výpočet detekcie hrán trval 1,87 a 3,86 ms v závislosti na rozmeroch snímky a výpočet priesečníkov trval v približne 2,9 ms a pri zmene rozmeru snímky sa takmer nezmenil. Trvanie posledných dvoch fáz je 0,6 ms pre výber hrán pre Y-spoj a 0,16 ms pre zostavenie šesťuholníkov. Aj tieto dve fázy nie sú závisle na rozmeroch snímky.

V kapitole 2.4 bol popísaný detektor využívajúci k detekcii Y-spoje. Pri rozmeroch snímky 1680x1120 trval výpočet nad čistým snímkom bez rušivých elementov 16,68 ms. K porovnaniu použijeme priemerný čas výpočtu pri spracovaní videa `rotation_FullHD.avi`. Jeho rozmery sú 1920x1080, čo je približne 1,1 – násobok oproti porovnávanému detektoru. Výsledný čas bol 6,76 ms. Tieto časy nemôžeme priamo porovnať, pretože detektory neboli testované na rovnakých zostavách. Môžeme však predpokladať, že algoritmus vytvorený v tejto práci dosahuje lepšieho výkonu, keďže bol testovaný na menej výkonnom procesore.

Test vplyvu zložitosti snímky

Druhým testom budeme zisťovať časy výpočtu jednotlivých fáz algoritmu v závislosti na zložitosti spracovávanej snímky. K tomuto účelu bolo vytvorené video `occlusion`. Použijeme rozlíšenie HD, teda 1280 x 720. Na začiatku videa je obrazovka prekrytá bielym papierom, aby sme simulovali čo najjednoduchšiu situáciu pre detektor hrán, ktorý takto neobjaví žiadnu hranu. Ostatné fázy algoritmu budú v tomto prípade bez dát k spracovaniu. Papier je následne postupne odstraňovaný, pričom sa zvyšuje počet detekovaných hrán a následne aj priesečníkov atď. Výsledok testu nad 108 snímkovým videom je zobrazený v grafe 6.5.



Obrázok 6.5: Graf času výpočtu jednotlivých fáz algoritmu v závislosti na zložitosti vstupnej snímky.

Prvých 21 snímiek videa je plne prekrytých a prvá fáza detekuje 0 hrán. Stále pritom algoritmus detekcie hrán zaberá v priemere 1,5 ms. Z toho môžeme vyvodiť, že samotné použitie skenovacích čiar potrebuje celých 1,5 ms. Môžeme preto zistiť čas potrebný na aproximáciu hrany, tak že od priemerného času prvej fázy algoritmu na konci videa odčítame 1,5 ms. Výsledný čas aproximácie hrán pre plne odkrytú snímku je $(2,0-1,5)ms = 0.5ms$.

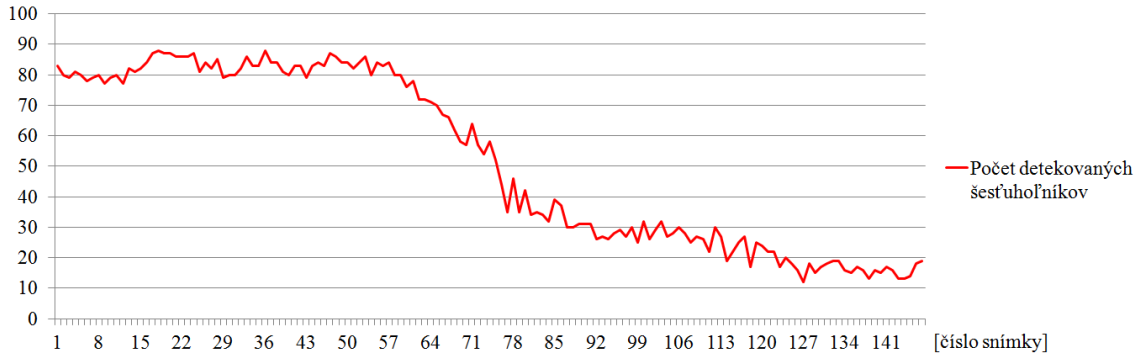
Po odkrytí prvých hrán sa postupne zvyšuje počet spracovávaných dát vo všetkých fázach algoritmu. Približne od snímky 85 sa časy jednotlivých fáz začínajú ustaľovať, pričom vo videu v tomto čase sú odkryté tri štvrtiny poľa šesťuholníkov.

Oscilovanie čiar v grafe spôsobujú zmeny v obraze na úrovni pixelov. Pri každom pohybe kamery sa totiž hrana detekuje na inom mieste s mierne odlišnou normálou.

6.3 Testy úspešnosti detekcie

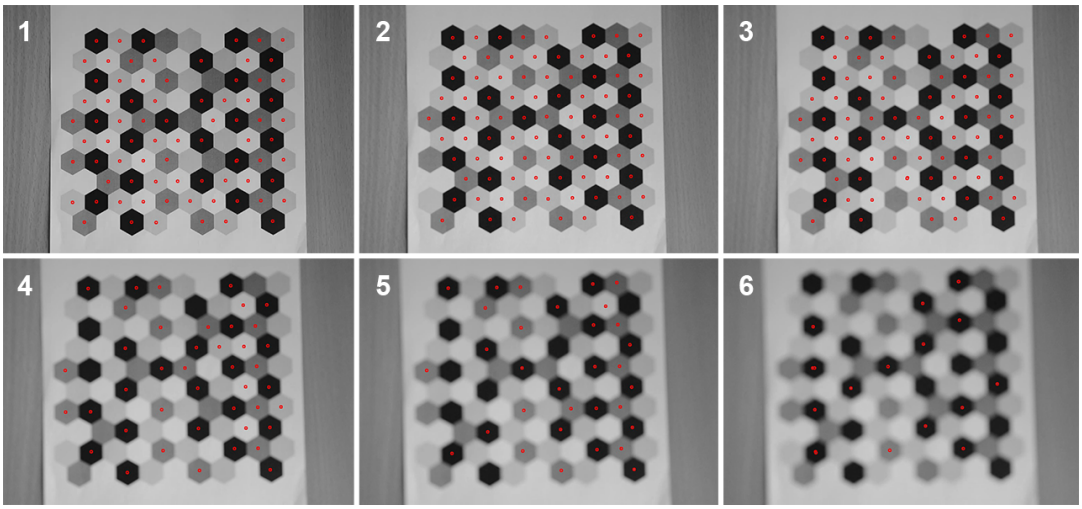
Test vplyvu rozostrenia snímky a rozmazania pohybom

Za účelom testovania odolnosti algoritmu voči rozmazaniu obrazu bolo vytvorených 10 snímiek. Šesť z nich je vytvorených použitím postupného rozostrovania obrazu pomocou manuálneho zaostrovania na objektíve digitálnej zrkadlovky. Zvyšné štyri snímky vznikli pri rôznych rýchlostiach pohybu zrkadlovky nad poľom markerov. Tieto snímky boli vybrané z dvoch videí `blur_HD.avi` a `motionBlur_HD.avi` a reprezentujú rôzne miery rozmazania.



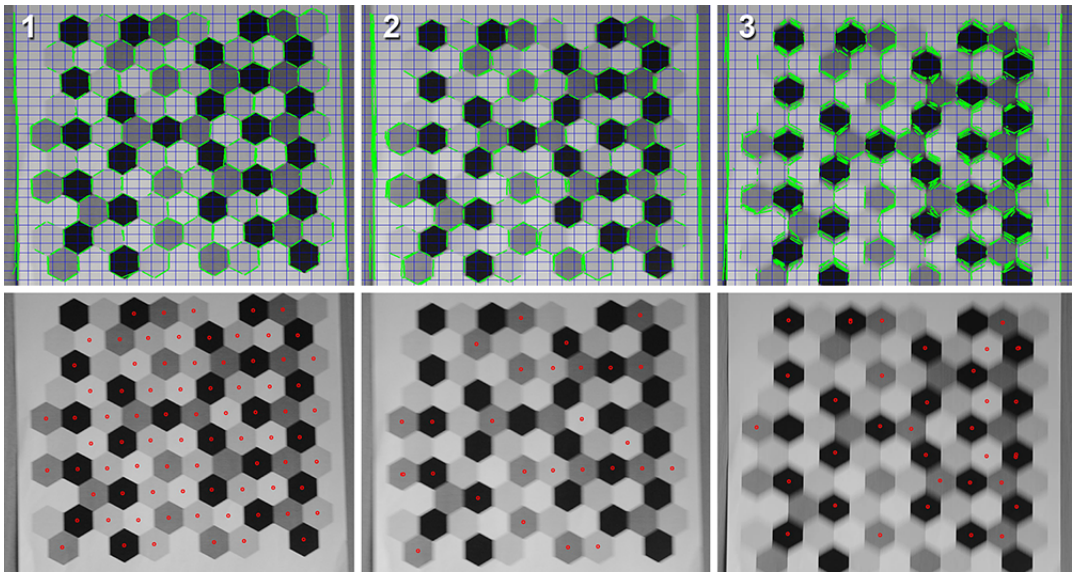
Obrázok 6.6: Graf počtu zdetekovaných šesťuholníkov v závislosti na rozostrení vstupnej snímky.

Z grafu 6.6 môžeme označiť snímku 60 ako zlomový bod, kedy sa počet zdetekovaných šesťuholníkov začína zmenšovať. Prudšia zmena nastáva približne v snímke 73, kde sa počet detekcií znížil o 20%. Na obrázku 6.7 sú zobrazené snímky zachytávajúce stav rozmazania snímky a stav detekcie šesťuholníkov vo videu `blur_HD.avi`.



Obrázok 6.7: Snímky s rôznym rozostrením odpovedajúce snímkam videa `blur_HD.avi`. Obr. č 1) odpovedá snímke 1; 2) snímke 55; 3) snímke 75; 4) snímke 90; 5) snímke 120 a 6) snímke 145

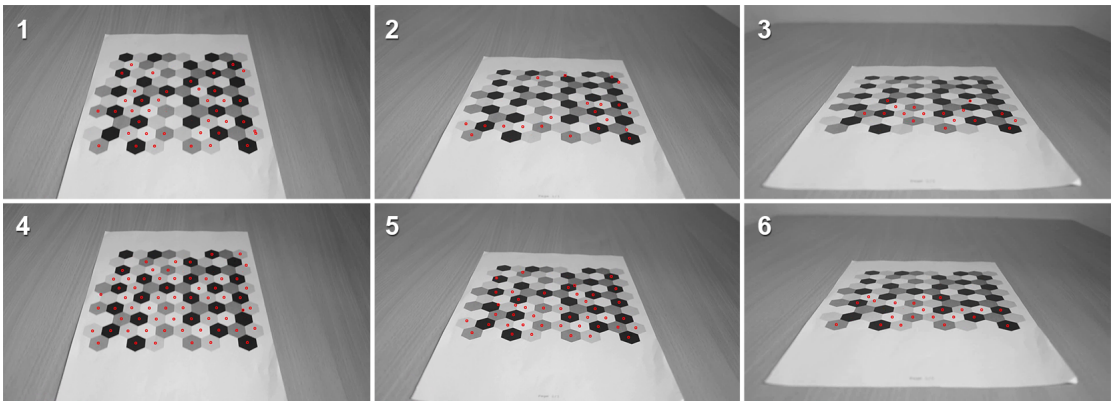
Aj pri podstatnom rozmazaní, aké je zobrazené obrázku 6.7 č. 4, je algoritmus schopný zdetekovať takmer polovicu šesťuholníkov. Väčšie rozmazanie už spôsobuje veľké poškodenie informácie o polohe hrán a pozície šesťuholníkov.



Obrázok 6.8: Snímky s rôznym rozmazaným pohybom. Prvý rad obrázkov zobrazuje detekované hrany (zelené). Druhý rad zobrazuje detekované šesťuholníky.

Test vplyvu snímania pod uhlom

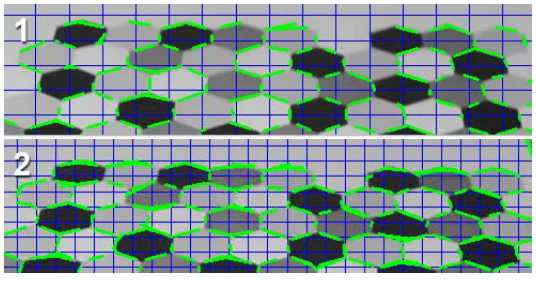
Nasledujúcim testom zistíme schopnosť algoritmu detekovať markery na naklonenej ploche. Použijeme k tomu video `perspective_HD.avi`. Na tomto videu je pole markerov snímané kolmo na plochu s markermi a postupne sa kamera posúva po oblúku k smerom k ploche. Do úvahy budeme brať len prvých 220 snímiek, kedy kamera zmení uhol z 90 na približne 15 stupňov.



Obrázok 6.9: Pole markerov pod rôznym uhlom pri počte skenovacích čiar 40 x 30 (obr. 1-3) a 70 x 50 (obr. 4-6). Približný uhol pohľadu pre snímky vľavo je 45 stupňov, v strede 30 stupňov a vpravo 20 stupňov.

Po prvom teste bol zo snímky zobrazujúcej detekované hrany (obr. 6.10) pozorovateľný nedostatočný počet nájdených hrán potrebných na definovanie tvaru šesťuholníka. Preto bolo množstvo hrán zvýšené zo 40x30 na 70x50. Počet detekcií hrán sa zvýšil, avšak kvalita aproximácie hrany je dosť nepresná pri nízkom rozlíšení použitého videa. Detekované hrany sú častejšie odklonené od odpovedajúcich hrán v obraze. Narástla preto aj pravdepodobnosť vyhľadania chybného šesťuholníka.

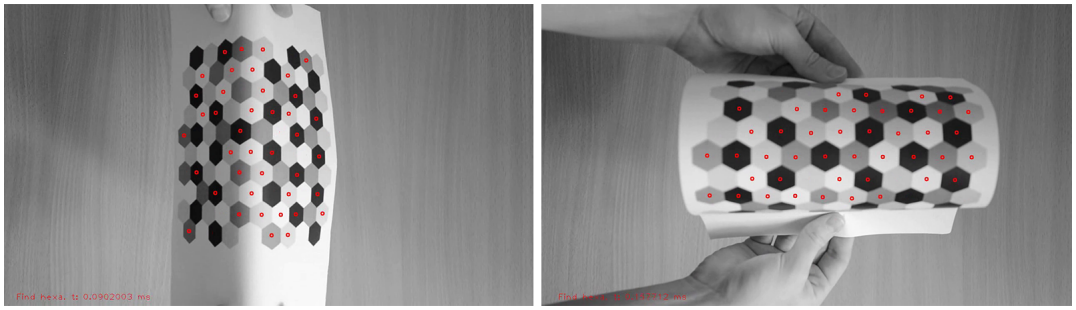
Všeobecný problém implementovaného algoritmu je slabá úspešnosť detekcie pri malých rozmeroch vstupného obrazu. Na rozdiel od algoritmov pracujúcich s planárnymi poľami markerov sa nemôžeme spoľahnúť na doplnenie časti geometrie markeru pomocou vytvorenia siete z čiastkových informácií. Každý marker je vyhľadávaný samostatne, čím sa podstatne zvyšuje pravdepodobnosť nedostatočného množstva informácií alebo chybnej detekcie.



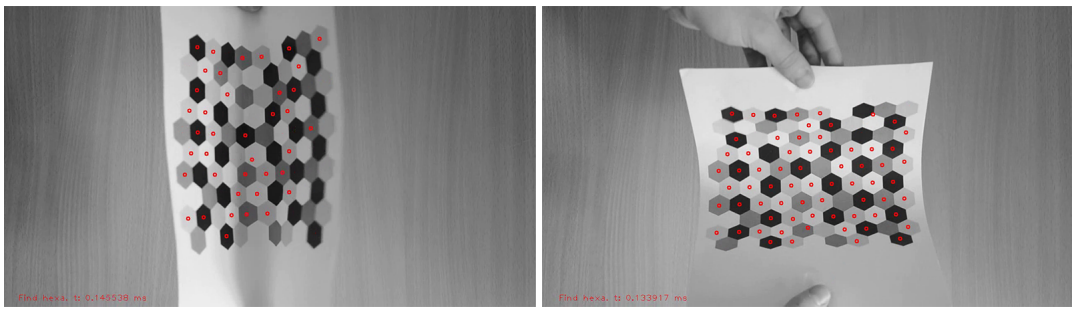
Obrázok 6.10: Detail na časť poľa markerov pri počte skenovacích čiar 40x30 (snímka 1) a 70x50 (snímka 2).

Test odolnosti voči deformácii

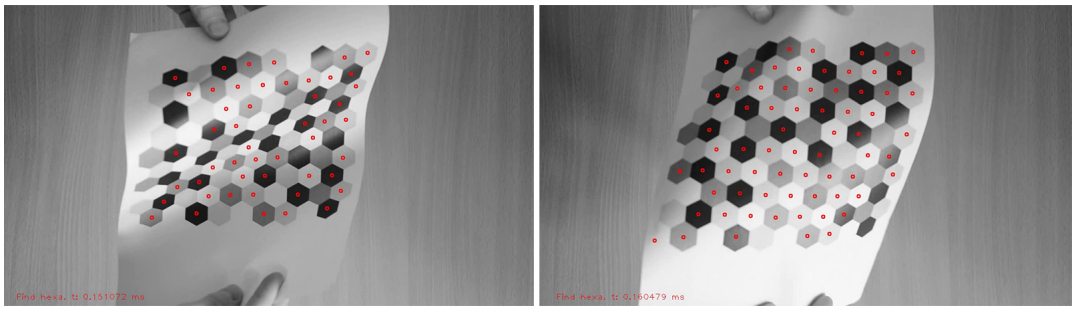
Odolnosť voči deformácii budeme testovať na videu `deformation_HD`. Papier s poľom markerov bol deformovaný na oboch osách a nakoniec bol zvlnený diagonálne. Pri poslednom teste je pole markerov na obr. 6.13 zároveň vystavené nerovnomernému osvetleniu. Vo väčšine uvedených obrázkov algoritmus detekoval viac ako polovicu šesťuholníkov.



Obrázok 6.11: Deformácia stredu poľa markerov do popredia.



Obrázok 6.12: Deformácia stredu poľa markerov do úzadia.



Obrázok 6.13: Deformácia poľa markerov zvlnením.

6.4 Test rýchlosti CUDA implementácie

Výkon algoritmu bol meraný dvomi spôsobmi. Prvé meranie bolo vykonané s použitím profilovacieho nástroja NSight, ktorý je rozšírením pre MS Visual Studio. Tento nástroj umožňil

zmerať časy vykonávania jednotlivých kernelov. Výsledky sú v tabuľke 6.1. Spracovaný bol obrázok rozmerov 1280x720.

Kernel	Čas (μs)
findEdges_Hor_D<uint=128>	102,080
findEdges_Vert_D<uint=128>	137,472
reorderEdges_D	6,112
aproximateEdge_D	166,400

Tabuľka 6.1: Časová náročnosť jednotlivých kernelov.

Všetky kernely uvedené v tabuľke 6.1 boli vytvorené špeciálne pre účel detekcie a aproximácie hrany. Navyše sa však pri výpočte používa aj paralelná suma prefixov z knižnice `Thrust`. Pri meraní časov pomocou systémových funkcií sa zistilo, že práve suma prefixov dramaticky znižuje výkon celého algoritmu. Pri analýze časov na videu v rozlíšení 1280x720 trvalo vykonanie sumy prefixov približne 2 milisekundy, pričom ostatné kernely boli vyhodnotené za približne 0,4 milisekundy.

Pokiaľ by sa hodnotili jednotlivé časti algoritmu zvlášť, tak použitie GPU na vyhľadanie a aproximáciu hrán urýchlilo výpočet oproti CPU verzii. Avšak súčasťou algoritmu je aj suma prefixov a ak pripočítame jej čas výpočtu, výsledné urýchlenie výpočtu v porovnaní s CPU algoritmom sa anuluje.

Návrh verzie algoritmu na GPU môžeme označiť ako čiastočne úspešný. Jeho najväčším nedostatkom je nutnosť použitia sumy prefixov, ktorá v porovnaní s ostnými kernelmi zaberá 5x viac času na výpočet.

Kapitola 7

Záver

V teoretickej časti projektu sme sa zoznámili s pojmom rozšírená realita a venovali sme sa technikám, ktoré sa používajú na sledovanie okolia systému rozšírenej reality. Po oboznámení sa s klasickými markermi boli podrobne naštudované Uniform Marker Fields, vrátane postupu použitého na detekciu šachovnicových UMF. Na ne nadviazala práca, v ktorej bolo uvedené pole markerov v tvare včelieho plástu. Prezentovaný detektor využíval k detekcii takzvané Y-spoje.

Poznatky užitočné pre vytvorenie optimalizovanej paralelnej verzie algoritmu pre rozšírenú realitu boli získané štúdiom platformy CUDA.

Pre modifikáciu pôvodného UMF bol navrhnutý postup detekcie šesťuholníkov na procesore.

CPU verzia algoritmu bola rozdelená na päť základných fáz. Vytvorené boli návrhy hlavných fáz pre platformu CUDA a analyzovaná bola ich miera využitia paralelného spracovania. CPU verzia bola plne implementovaná a na CUDA platformu bola vytvorená prvá časť detektora. Pomocou testovania sme zistili optimálnu hustotu skenovacích čiar. Porovnali sme časy výpočtu jednotlivých fáz algoritmu. Výsledná implementácia dosahuje vyššieho výkonu v porovnaní s tým, ktorý je potrebný k výpočtu v reálnom čase. Zistili sme limitujúce faktory a situácie, pri ktorých detektor stráca presnosť. Dokázali sme, že výsledný detektor je schopný detekovať šesťuholníky na deformovanom poli markerov.

Implementácia na platforme CUDA dosiahla podstatné zrýchlenie u častí algoritmu zaisťujúcich detekciu a aproximáciu hrany, avšak použitie funkcie sumy prefixov z externej knižnice anulovalo dosiahnuté zrýchlenie.

Ďalší vývoj by sa mohol uberať smerom dodatočného dohľadávania hrán na miestach, kde s vysokou pravdepodobnosťou došlo k výpadku hrany. Taktiež by mohol byť detektor rozšírený o dynamickú úpravu parametrov detekcie počas spracovávania jednotlivých snímiek.

Literatura

[1] Fua, P.; Lepetit, V.: Vision Based 3D Tracking and Pose Estimation for Mixed Reality.
URL http://www.igi-pub.com/downloads/excerpts/1599040689ch1.pdf

[2] Gregory Kipper, Joseph Rampolla: *Augmented Reality*. Elsevier, 2013, iSBN 978-1-59749-733-6.

[3] Harris, M.: Parallel Prefix Sum (Scan) with CUDA. [online], [prevzaté 2013-05-13].
URL http://beowulf.lcs.mit.edu/18.337-2008/lectslides/scan.pdf

[4] Herout, A.; Szentandrási, I.; Zachariáš, M.; aj.: Five Shades of Grey for Fast and Reliable Camera Pose Estimation. In *Proceedings of CVPR*, 2013.

[5] Horváth, Z.; Herout, A.; Szentandrási, I.; aj.: Design and Detection of Local Geometric Features for Deformable Marker Fields. In *Proceedings of 29th Spring conference on Computer Graphics*, Comenius University in Bratislava, 2013, ISBN 978-80-223-3377-1, s. 85–92.
URL http://www.fit.vutbr.cz/research/view_pub.php?id=10344

[6] Kato, H.; Billinghurst, M.: Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System. In *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, IWAR '99, Washington, DC, USA: IEEE Computer Society, 1999, ISBN 0-7695-0359-4, s. 85–.
URL http://dl.acm.org/citation.cfm?id=857202.858134

[7] Naimark, L.; Foxlin, E.: Circular Data Matrix Fiducial System and Robust Image Processing for a Wearable Vision-Inertial Self-Tracker. In *Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, ISMAR '02, Washington, DC, USA: IEEE Computer Society, 2002, ISBN 0-7695-1781-1, s. 27–.
URL http://dl.acm.org/citation.cfm?id=850976.854961

[8] NVIDIA Corporation: CUDA C PROGRAMMING GUIDE. [online], [prevzaté 2013-05-13].
URL http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

[9] NVIDIA Corporation: THRUST QUICK START GUIDE. [online], [prevzaté 2013-05-13].
URL http://docs.nvidia.com/cuda/pdf/Thrust_Quick_Start_Guide.pdf

[10] Reiners, D.; Stricker, D.; Klinker, G.; aj.: Augmented Reality for Construction Tasks: Doorlock Assembly. In *IN PROC. IEEE AND ACM IWAR98 (1. INTERNATIONAL WORKSHOP ON AUGMENTED REALITY*, AK Peters, 1998, s. 31–46.

[11] Rekimoto, J.: Matrix: a realtime object identification and registration method for augmented reality. In *Computer Human Interaction, 1998. Proceedings. 3rd Asia Pacific*, jul 1998, s. 63 –68, doi:10.1109/APCHI.1998.704151.

[12] Szentandrási, I.; Zachariáš, M.; Havel, J.; aj.: Uniform Marker Fields: Camera Localization By Orientable De Bruijn Tori. In *Proceedings of ISMAR*, 2012.

[13] Zhang, X.; Fronz, S.; Navab, N.: Visual Marker Detection and Decoding in AR Systems: A Comparative Study. In *Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, ISMAR '02, Washington, DC, USA: IEEE Computer Society, 2002, ISBN 0-7695-1781-1, s. 97–.
URL http://dl.acm.org/citation.cfm?id=850976.854955

[14] Zhou, F.; Duh, H.-L.; Billinghurst, M.: Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. sept. 2008: s. 193 –202, doi:10.1109/ISMAR.2008.4637362.

Príloha A

Obsah DVD

- `/`
  - `doc` Technická správa a prílohy.
    - `doc.pdf` Text technickej správy.
  - `src` Zdrojové súbory.
    - `doc` Technická správa.
    - `markers` Zdrojové súbory detektoru, predpripravené testovacie dávky a dáta k testovaniu.
      - `image` Snímky použité pri testovaní.
      - `video` Videá použité pri testovaní.
      - `source` Zdrojové súbory a projekt MS Visual Studia.
  - `plagat.png`
  - `video.mp4`

Príloha B

Ovládanie programu

Pri spustení aplikácie je potrebné uviesť cesty cestu k aspoň jednému obrázku alebo videu. Pomocou nasledujúcich argumentov sa programu na mieste <input> zadávajú potrebné informácie.

-v=<string>
slúži na zadanie cesty k videu vo formáte `avi`
-i=<string>
slúži na zadanie cesty k obrázku vo formáte `jpg, png`

-vl=<number>
nastavenie počtu vertikálnych skenovacích čiar
-hl=<number>
nastavenie počtu horizontálnych skenovacích čiar
-me=<number>
nastavenie maximálneho počtu hrán na jeden snímok (definuje veľkosť alokovaného poľa)

-gw=<number>
šírka mriežky pre uloženie hrán
-gh=<number>
výška mriežky pre uloženie hrán

Program dokáže pracovať s viacerými videami a snímkami zadanými cez príkazový riadok. Ak bude príkazový riadok obsahovať cesty ku trom snímkam a k piatim videám, program po spustení zobrazí výber, kde číslom 0-2 vyberieme jednu zo snímiek a jedno z videí vyberieme písmenom a-e. Po ukončení spracovania sa vráti spať na túto obrazovku a je možné si znova vybrať.

Výsledky spracovania sú ukladané do priečinku s názvom rovnakým ako vstupné video či snímka. Do tohto priečinku sa ukladajú aj štatistiky.